공학박사학위논문

# An Investigation of Incremental Hybrid Genetic Algorithm For Subgraph Isomorphism Problem

## 부분 그래프 동형 사상 문제를 위한 점진적 유전 알고리즘의 조사

2019년 2월

서울대학교 대학원

전기·컴퓨터공학부

최 혁 근

# An Investigation of Incremental Hybrid Genetic Algorithm For Subgraph Isomorphism Problem

지도교수 문 병 로

이 논문을 공학박사 학위논문으로 제출함

2018년 11월

서울대학교 대학원
전기·컴퓨터공학부
최 혁 근

최 혁 근의 공학박사 학위논문을 인준함
2018년 12월

| | | |
|---|---|---|
| 위 원 장 | 신 영 길 | (인) |
| 부위원장 | 문 병 로 | (인) |
| 위 원 | 엄 현 상 | (인) |
| 위 원 | 김 용 혁 | (인) |
| 위 원 | 정 순 철 | (인) |

# An Investigation of Incremental Hybrid Genetic Algorithm For Subgraph Isomorphism Problem

by

HyukGeun Choi

Department of Electrical Engineering & Computer Science
Seoul National University
2019

## Abstract

Graph is the most representative data structure for modeling the relationships of objects and graph pattern matching is one of the key problems that arise in many applications where data is expressed in the form of graph. Although graph pattern matching can be defined by a semantic-based method using information such as the labels of vertices or edges, it is generally defined by a structure-based method using only the relationships between vertices and edges, and such pattern matching is represented by the subgraph isomorphism.

The algorithms proposed so far to solve the subgraph isomorphism problem are classified into two types. The first is an exact method to find out all existing solutions based on the recursive backtracking algorithm. However, since the subgraph isomorphism problem is NP-Complete, if all the permutations are searched one by one, the running time increases exponen-

tially according to the size of the problem. The second is an approximation method based on metaheuristic such as genetic algorithm. They are able to find good quality solutions within a reasonable amount of time, but most algorithms do not have enough search capability to cover the large and complex problem space of this problem.

The search capability of a metaheuristic algorithm can be improved by designing better operators or local heuristics, but it is possible to improve the performance greatly by changing the fitness function and by reforming the search strategy. If the original problem is divided into subproblems with the suitable size for the search capability of a metaheuristic algorithm, and each subproblem is solved step by step, the problem can be solved more efficiently. Also, if we change the fitness function to transform the fitness landscape more convex, the effect of an incremental algorithm will be much greater.

In this thesis, we propose an efficient incremental hybrid genetic algorithm to solve the subgraph isomorphism problem. First, we introduce a new fitness function which is suitable for the problem of the subgraph isomorphism problem and examine how the fitness landscape generated with the operator is transformed. We introduce a multi-objective fitness function by designing a new function reflecting the degree constraint of the subgraph isomorphism. Through the experiments, we analyze the characteristics of the new fitness function combining with the local optimization algorithm, investigate the correlation between the fitness value and the average distance of the local optima to explain how the new fitness function transforms the fitness landscape of the subgraph isomorphism problem. We compare the re-

sults of the hybrid genetic algorithm applying the proposed multi-objective fitness function with that of the conventional genetic algorithm and show how the proposed fitness function facilitates the search capability of a genetic algorithm.

Second, we introduce the new efficient search strategy, the incremental genetic algorithm, and how the design issues are reflected in the process and performance of the algorithm. We divide the original problem into a sequence of successive subproblems with the optimal substructure, solve each subproblem through the hybrid genetic algorithm, and then extend the solutions obtained for the initial solutions of the next subproblem. This process is applied sequentially to develop the solutions of the small problem to those of the original problem. Through the experiments, we discuss how to divide the original problem into successive subproblems and analyze how components of the sequence affect the performance of the incremental genetic algorithm. We also compare the performance of the incremental hybrid genetic algorithm with that of the previous hybrid genetic algorithm through the random graph instances, and show a good scalability the proposed algorithm through the experimental results obtained for real data with a large size that was impossible with existing algorithms.

**Keywords :** Combinatorial Optimization, Fitness Landscape, Genetic Algorithm, Incremental Genetic Algorithm, Search Operator, Subgraph Isomorphism Problem

**Student Number :** 2009-20913

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

Graph is a simple and universal data representation to model pairwise relationships among a set of objects. One of the interesting problems encountered when handling graph data is a graph pattern matching arising from pattern recognition, knowledge discovery, biology, cheminformatics, dynamic network traffic and intelligence analysis [AW10, CFSV04a, Gal06, SWG02].

Given two graphs $G$ and $H$, the subgraph isomorphism problem is to determine whether $H$ contains a subgraph that is isomorphic to $G$ and this decision problem is well-known NP-Complete [Coo71]. Many algorithms have been proposed to solve this problem starting with the backtracking algorithm by Ullmann [Ull76]. VF2 [CFSV04b], QuickSI [SZLY08], GraphQL [HS08], GADDI [ZLY09] and SPath [ZH10] improved the performance by exploiting different join orders, pruning rules, and auxiliary information from the Ullmann algorithm [LHKL12]. The subgraph isomorphism problem is generalized to the maximum common subgraph problem, which finds the largest subgraph of two given graphs that are isomorphic to each other. The algorithms by McGregor's [McG82], Durand and Pasari's [DPBT99], Balas and Yu's [BY86] are representative ones for that prob-

lem. However these algorithms for both problems have exponential time complexity, their scalability are limited and they only work with auxiliary information such as vertex or edge labels.

Genetic algorithm also has been applied to solve this problem [BJWG94, WG94, FKZ01, FC15, ZWL+11, LCC16], but they showed limitations in terms of performance and scalability. In recent years, however, the performance of genetic algorithm for the subgraph isomorphism problem has been improved by designing a new fitness function [CYM12] and adopting proper searching techniques [CKM14]. And real world applications of the subgraph isomorphism have also been researched to which genetic algorithms have been successfully applied [KM10, KCYM16].

Existing backtracking algorithms based on Ullmann's algorithm look up only one vertex at a time. They can find the exact solution but search the problem space search inefficiently. On the contrary, the previous genetic algorithms handle a relatively large problem space at once, which leads to insufficient solution quality and limited scalability. If we design the appropriate problem space to fit the performance of a genetic algorithm at once and extend the obtained solutions one by one, the solution of the original problem can be found more efficiently.

## 1.2 Contribution

In this thesis, we design and analyze the incremental hybrid genetic algorithm for the subgraph isomorphism problem to implement more advanced algorithm.

First, we inspect a multi-objective fitness function that is built by combining a new term based on the degree constraint to the fitness function commonly used. With a two exchange local heuristic, we investigate the difference of fitness landscape generated by a single objective and a multi objective function and compare the performance of the hybrid genetic algorithm applying the new fitness function with that of conventional genetic algorithm.

After then, the fundamentals and design issues of incremental genetic algorithm are introduced. Incremental genetic algorithm begins with decomposing the original problem into a sequence of consecutive subproblems based on the optimal substructure. And the solutions to one subproblem found by a hybrid genetic algorithm are extended and are used as initial solutions of the next subproblem. By applying this process sequentially, the high quality solutions of the original problem can be obtained. In the incremental process, the design of a sequence of subproblems and the search capability of a genetic algorithm critically influence the overall performance. We focus on three design issues: vertex reordering to determine the overall search path, problem expansion size to determine a coverage of problem space to be searched for each subproblem and stopping criterion to maintain the diversity of solutions. In addition, a multi-objective fitness function and genetic operators are designed to improve the performance of the hybrid genetic algorithm.

The effects of design schemes are analyzed on synthetic dataset and it is figured out what is important to maintain the diversity of solutions during the incremental process. Based on the analysis, we show that the incremen-

tal approach with appropriate design schemes significantly improve the performance. Furthermore, the proposed algorithm is tested using real world dataset with large size graphs. From the experiments with two datasets, we will show that the incremental hybrid genetic algorithm performs better and has more scalability than previous studies.

Some portions of the work discussed in this thesis have been based on [KM10, CYM12, CKM14, KCYM16].

## 1.3 Organization

The thesis is organized as follows: Chapter 2 provides the introduction to subgraph isomorphism and its related problems. The review of the basic concepts and the structures of genetic algorithm is also presented. In Chapter 3, we begin with reasoning out the limitation of the previous genetic algorithms for the subgraph isomorphism problem. And then, we provide an analysis of the degree constraint of the subgraph isomorphism and combine this term into the existing fitness function. The effectiveness of the new multi-objective fitness function is tested in the various experiments. In Chapter 4, the incremental process and related design issues are described. The experimental results are also provided for the design scheme in detail and the performance of the incremental hybrid genetic algorithm discussions is measured for synthetic and realworld dataset. Conclusions are in Chapter 5.

# Chapter 2

# Preliminary

## 2.1 Graph Pattern Matching and Isomorphism

How do we determine that two graphs are similar? To be precise, how do we determine that two graph are structurally similar even though they differ in appearance? These questions are formally expressed by the notion of *graph isomorphism* or simply *isomorphism*.

**Definition 1** (Graph Isomorphism)**.** *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be graphs. An isomorphism of two graphs G and H is a bijection between the vertex sets of G and H, $g : V_G \to V_H$, such that $(u,v) \in E_G$ if and only if $(g(u), g(v)) \in E_H$.*

This kind of bijection is generally called edge-preserving bijection, in accordance with the general notion of *isomorphism* being a structure-preserving bijection. If an *isomorphism* exists between two graphs $G$ and $H$, then the graphs are called *isomorphic* to each other and we write $G \simeq H$.

*Graph isomorphism* is an equivalence relation on graphs and as such it partitions the class of all graphs into equivalence classes. A set of graphs *isomorphic* to each other is called an isomorphism class of graphs. It means that *isomorphism* allows us to distinguish graph properties inherent to the structures of graphs themselves from properties associated with graph rep-

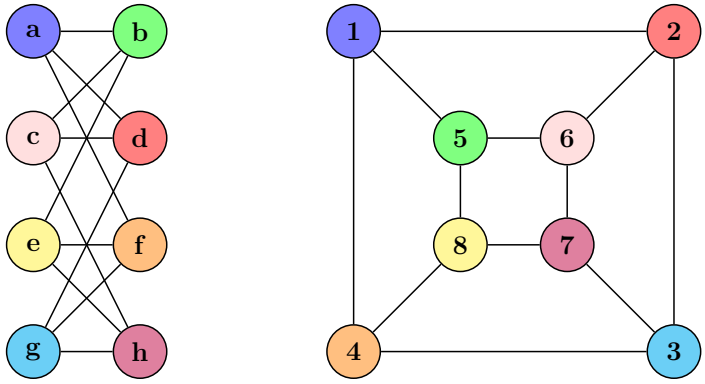



Figure 1: The example of two graph that is isomorphic to each other.

resentations: graph drawings, data structures for graphs, graph labelings, etc. For example, if a graph has exactly one cycle, then all graphs in its isomorphism class also have exactly one cycle. On the other hand, in the common case when the vertices of a graph are (represented by) the integers $1,2,\ldots,N$, then the expression $\sum_{v\in V(G)} v\cdot \deg v$ may be different for two isomorphic graphs.

Furthermore, detecting patterns in data is a common task that needs to be accomplished often in practice and this corresponds to finding a copy of some previously described smaller graph. It is related to the *subgraph isomorphism* that is a generalization of *graph isomorphism*, which is an important and general form of exact pattern matching arising from various applications.

**Definition 2** (Subgraph Isomorphism)**.** *Let $G=(V_G,E_G)$ and $H=(V_H,E_H)$ be graphs. A subgraph isomorphism from $G$ to $H$ is a injective function $g: V_G\to V_H$ such that $(u,v)\in E_G$, then $(g(u),g(v))\in E_H$. $g$ is called an induced subgraph isomorphism if in addition if $(u,v)\notin E_G$, then $(g(u),g(v))\notin E_H$.*

If there is a *subgraph isomorphism* from $G$ to $H$, we say that a graph $G$ is *isomorphic* to a subgraph $S$ of a graph $H$, denoted by $G \simeq S \subseteq H$. The difference between *subgraph isomorphism* and *induced subgraph isomorphism* is that, in *induced subgraph isomorphism*, the absence of an edge in $G$ implies that the corresponding edge in $H$ must also be absent.

## 2.2 Subgraph Isomorphism and Related Problems

There are several pattern matching problems related to *subgraph isomorphism*.

- *Graph Isomorphism Problem*

  Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, the *graph isomorphism problem* is to determine whether $G$ is isomorphic to $H$. If two graphs are isomorphic to each other, the number of vertices and edges in both graphs must be the same. The *graph isomorphism problem* is one of few standard problems in computational complexity theory belonging to NP, but not known to belong to either of its well-known (and, if $P \neq NP$, disjoint) subsets: P and NP-complete. It is one of only two, out of 12 total, problems listed [GJ02] whose complexity remains unresolved, the other being integer factorization. It is however known that if the problem is NP-complete then the polynomial hierarchy collapses to a finite level [Sch88].

- *Subgraph isomorphism problem*

Given two graphs $G = (V_G, E_H)$ and $H = (V_H, E_H)$, the *subgraph isomorphism problem* is to determine whether there exists a subgraph $S \subseteq H$ such that $g : V_G \to V_H$ is a isomorphism from $G$ to $S$. This decision problem is a well-known NP-Complete problem [Coo71]. The *subgraph isomorphism problem* even in the planar graphs also remains NP-complete, because the *Hamiltonian cycle problem* for planar graphs is NP-complete [dlHJ+13].

In the context of the Aanderaa–Karp–Rosenberg conjecture on the query complexity of monotone graph properties, Gröger [Die92] showed that any subgraph isomorphism problem has query complexity $\Omega(n^{3/2})$; that is, solving the *subgraph isomorphism* requires an algorithm to check the presence or absence in the input of $\Omega(n^{3/2})$ different edges in the graph.

- *Maximum common subgraph problem*

  Given two graphs $G = (V_G, E_H)$ and $H = (V_H, E_H)$, and positive integer $K$, the *maximum common subgraph problem* is to determine whether there exist subsets $E_{S_G} \subseteq E_G$ and $E_{S_H} \subseteq E_H$ with $|E_{S_G}| = |E_{S_H}| \leq K$ such that the two subgraphs $S_G = (V_{S_G}, E_{S_G})$ and $S_H = (V_{S_H}, E_{S_H})$ are isomorphic.

  A common subgraph of two graphs $G$ and $H$ consists of a subgraph $S_G$ of $G$ and a subgraph $S_H$ of $H$ such that $S_G \simeq S_H$. The maximum common subgraph is the largest possible common subgraph, where a common subgraph is maximal if it cannot be extended to another common subgraph by the addition of vertices or edge.

This problem is a generalization of the subgraph isomorphism problem, and also known as NP-complete.

- Graph edit distance problem

  Given a set of graph edit operations, the graph edit distance between two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, written as $GED(G,H)$ can be defined as

$$GED(G,H) = \min_{(e_1,e_2\ldots,e_k)} \in \mathcal{P}(G,H) \sum_{i=1}^{k} c(e_i)$$

  where $\mathcal{P}(G,H)$ denotes the set of edit paths transforming $G$ into (a graph isomorphic to) $H$, and $c(e) \geq 0$ is the cost of each graph edit operation $e$. The set of elementary graph edit operators typically includes vertex insertion, vertex deletion, vertex substitution, edge insertion, edge deletion, edge substitution. Given two graphs and the positive integer $K$, the *graph edit distance problem* is to determine whether there exists $GED(G,H)$ such that $GED(G,H) \leq K$.

  The graph edit distance is a measure of similarity between two graphs and this concept is applied in inexact graph matching such as error-tolerant pattern recognition in machine learning [GXTL10].

## 2.3 Genetic Algorithm

Darwin's theory of evolution offers an explanation of the biological diversity and its underlying mechanisms. In what is sometimes called the macroscopic view of evolution, natural selection plays a central role. Given

an environment that can embrace only a limited number of individuals, and the basic instinct of individuals to reproduce, selection becomes inevitable if the population size is not to grow exponentially. Natural selection favors those individuals that compete for the given resources most effectively, in other words, those that are adapted or fit to the environmental conditions best, which is well-known as survival of the fittest. Competition-based selection is one of the two foundation of evolutionary process. The other primary force results from phenotypic variations among individuals of the population. Phenotype are those behavioral and physical features of an individual, so that it determine the fitness that directly affect its response to the environment. Each individual represents an unique combination of phenotypic traits that is evaluated by the environment. If it evaluates with favor, then it is propagated via the individual's offspring, otherwise it is discarded without offsprings. Darwin's insight was that small, random variations in phenotype occur during reproduction from generation to generation. Through these mutations, new combination of phenotype occur and get evaluated.

The genetic algorithm (GA) captures this basic model through a population of individuals. These individuals are the units of selection, that is to say that their reproduce, mutate give rise to new individuals to be evaluated. Thus, as time passes, there is a change in the constitution of the population, i.e., the population is the unit of evolution [ES$^+$03].

Since Holland established the basic principles [Hol75], genetic algorithm has been developed and applied to many optimization problems in various fields including computer science, bioinformatics and chemistry.

### 2.3.1 Structure

The basic structure of genetic algorithm is as follows: Given a population of individuals, the environmental pressure causes natural selection, survival of the fittest, which causes a rise in the fitness of the population. Given a quality function to be minimized(or maximized), we can randomly create a set of candidates, i.e., elements of the function's domain, and apply the quality function as an abstract fitness measure. Based on this fitness, some of the better candidates are chosen to seed the next generation by applying recombination. Crossover is an operator applied to parents and results new candidates. Mutation is applied to one candidate and results in one new offspring. Executing crossover and mutation leads to a set of new offsprings that compete based on their fitness with the old ones in the population for a place in the next generation. This process can be iterated until an individual with sufficient quality is found or a previously set computational limit is reached.

The general scheme of an genetic algorithm in pseudocode is given in Algorithm 1. This scheme matches the category of generate-and-test algorithms. The fitness function represents a heuristic estimation of solution quality, and the search process is driven by the crossover, mutate and the selection operators. Genetic algorithm posses a number of features that can help to position them within in the family of generate-and-test methods:

- GA is population based, i.e., they process a whole collection of candidate solutions simultaneously.
- GA mostly uses recombination to mix information of more candidate

solutions into a new one.

- GA is stochastic.

**Algorithm 1:** The general scheme of a genetic algorithm

```
1: INITIALIZE population and EVALUATE
2: while termination condition is not satisfied do
3:     SELECT parents
4:     CROSSOVER pairs of parents
5:     MUTATE the resulting offspring
6:     EVALUATE new candidates
7:     REPLACE individuals for the next generation
8: end while
```

### 2.3.2 Representation

Objects forming possible solutions within the original problem context are referred to as phenotypes, while their encoding, that is, the individuals are called genotypes. Representation amounts to specifying a mapping from the phenotypes onto a set of genotypes that are said to represent these phenotypes. It is important to understand that the phenotype space can be vary different from the genotype space, and that the whole search takes place in the genotype space.

When choosing a representation, it is important to choose the proper representation for the problem being solved. Thus choosing the most appropriate representation for the problem is most important but difficult parts of designing a good genetic algorithm. Often this only comes with practice and a good knowledge of the application domain.

Many representations have been introduced since genetic algorithm

become popular. While binary string representation is the most frequently used, several encodings have used frequently such as integer encoding, real-value encoding, locus-based coding, etc. According to the No Free Lunch Theorem [WM97], there is no one encoding that outperform the others for all problems.

### 2.3.3 Fitness Function

The role of fitness function is to represent the requirements to adapt to. It forms the basis for selection, and thereby it is a key factor that enables improvements. More accurately, it defines what improvement means. From the problem-solving perspective, it represents the task to solve in the genetic process. Technically, it is a function or procedure that assigns a quality measure to genotypes. Typically, this function is composed of a quality measure in the phenotype space and the inverse representation. In fact, the fitness function is one of the important factors that determine the problem space in combination with the operators, and the ability to search a problem space is determined by how to design a fitness function and related search operators.

In this thesis, we examine the transformation of problem space and the change of the search ability of local heuristics by adding an auxiliary constraint to the fitness function.

### 2.3.4 Crossover

Crossover, the process whereby a new individual solution is created from the information contained within two or more parents, is considered

by many to be one of the most important features in genetic algorithm. The principal behind crossover is simple that by mating two individuals with different but desirable features, we can produce an offspring which combine both of those features. This principal has a strong supporting case that it is one which has been successfully applied for millennia by breeders of plants and livestock, to produce species which give higher yields or have other desirable feature. Genetic algorithm creates a number of offspring by random crossover, accepts that some will have undesirable combinations of traits, most may be no better or worse than their parents, and hopes that some have improved characteristics.

The ability to combine partial solutions via crossover is certainly one of the features that most distinguishes GA from other global optimization algorithms. Crossover operators are usually applied probabilistically according to a crossover rate $p_c$. Typically, two parents are selected and two offspring are created via crossover of the two parents with probability $p_c$; or by simply copying the parents, with probability $1 - p_c$. Crossover rate determines to control how parts of the chromosome are perturbed independently, so hat this process determines the chance that a chosen pair of parents undergoes this operator.

### 2.3.5 Mutation

Mutation is a unary variation operator that is applied to one genotype and deliver a stochastic modified mutant. This operation in genetic algorithm has not been so diverse as crossover. The most popular scheme is to choose each gene in a chromosome with uniform random probability and

flip it. Hybrid genetic algorithm usually require rather higher mutation rates because weakly perturbed chromosomes tend to return to the initial state when local search thus cannot escape local optima [VL91].

### 2.3.6 Hybrid Genetic Algorithm

Every optimization method has pros and cons. Though it is very effective method for global optimization, genetic algorithm is not an exception. Genetic algorithm is known to be weak in fine-tuning around local optima.

To compensate the problem, genetic algorithm has been combined with local optimization methods. This is called hybrid genetic algorithm or memetic algorithm.

Genetic operators in conventional GA have different purpose than those in hybrid GA. In conventional GA, crossover has to evolve better offsprings than parents, even though it happens with a low probability. On the other hand, the role of crossover in hybrid genetic algorithm is to explore a broad range of problem space. In other to take advantage of the performance of local optimization algorithm, it is aimed to provide an prospective initial solution to the local optimization algorithm.

# Chapter 3

# Inspecting Fitness Function of Subgraph Isomorphism Problem

In this chapter, we inspect the fitness function used in previous studies and introduce a multi-objective fitness function for the subgraph isomorphism problem.

First of all, the the previous studies for the subgraph isomorphism problem are surveyed. The subgraph isomorphism problem is redefined as the optimization problem by introducing a fitness function in order to apply a genetic algorithm. And then, we review the fitness function used in the previous studies and figure out that these functions do not fully reflect the property of the subgraph isomorphism. To compensate for the disadvantages of the previous function, we introduce a multi-objective fitness function function by adding a constraint to the existing fitness function. Lastely, through the experiments, it is shown that this fitness function transforms the fitness landscape more convex to explore effectively and the hybrid genetic algorithm using new fitness function shows better performance than that of using the previous fitness function.

## 3.1 Introduction

The first practical algorithm for the subgraph isomorphism problem was proposed in Ullmann's research [Ull76]. It is a recursive backtracking algorithm to find all subgraph isomorphisms between two graphs. After this study, several algorithms [CFSV04b, SZLY08, HS08, ZLY09, ZH10] have been proposed to enhance and to improve Ullmann's algorithm. These algorithms commonly improved the vertex join order and pruning rules to lop off infeasible candidates as early as possible. The subgraph isomorphism problem can be generalized to the maximum common subgraph problem, which finds the largest subgraph of two given graphs that are isomorphic to each other. The McGregor [McG82], Durand and Pasari [DPBT99] and Balas and Yu [BY86] proposed representative algorithms. However these algorithms have exponential time complexity, they have limited scalability or only work with auxiliary information such as vertex or edge labels.

In the previous, there were attempts to divide and reorganize this problem. Messmer and Bunke [MB00] introduced the method of decomposing model graphs into a set of subgraphs in advance. And then, when given the input graph, they recombined these small subgraphs into the complete subgraph isomorphism. They showed recombining the answers of small size problems recursively is helpful to build answers of the original problem.

The studies applying genetic algorithm are also proposed. In these studies, the fitness function was commonly defined as a number of edges that match or mismatch over the mapping and there have been various real world applications applying a genetic algorithm using this fitness function

into the subgraph isomorphism problem.

Brown et al. [BJWG94] applied it to 2D Chemical structure matching. Zhong et al. [ZWL+11] used it for the resource assignment in the real time digital simulator and Kim and Moon [KM10] proposed the malware detection system by solving the subgraph isomorphism problem with this fitness function.

Because these algorithms showed limitation in terms of performance or scalability, several studies were proposed to improve performance of a genetic algorithm for the subgraph isomorphism problem. Choi et al. [CYM12] introduced a multi-objective fitness function. They added the degree comparison result of the matched vertices between two graphs to the fitness function as a new constraint and showed that the fitness landscape generated by a multi-objective function is more globally convex than than that of a single objective function. After then, Choi et al. [CKM14] proposed a new hybrid genetic algorithm using an incremental process. Kim et al. [KCYM16] applied this methods to measure the source similarity and showed the similarity measure of this approach reflects the actual likeliness between the codes.

## 3.2 Conventional Fitness Function

In order to apply a genetic algorithm to the subgraph isomorphism problem, it is necessary to introduce a fitness function and the problem is converted into the form of an optimization problem, rather than addressing the decision problem of the subgraph isomorphism directly.

We describe the formal definition of the optimization problem for the subgraph isomorphism between two directed graphs.

**Definition 3** (Subgraph Isomorphism Problem)**.** *Given two directed graphs* $G = (V_G, E_G)$ *and* $H = (V_H, E_H)$ *where* $|V_G| \leq |V_H|$*, the subgraph isomorphism problem, denote by* $\boldsymbol{SIP}(G,H)$*, is to find an injective function* $g : V_G \rightarrow V_H$ *that minimizes the fitness function* $f$*. The optimal solution, with the fitness value of 0, is the subgraph isomorphism from* $G$ *to* $H$*.*

In previous studies, The fitness function has been commonly defined as a number of edges that match or mismatch over the mapping. Given a mapping $g : V_G \rightarrow V_H$, this function for minimization is formulated is as follows:

$$f_1(g) = \sum_{e \in E_G} I(e, E_S) + \sum_{e \in E_S} I(e, E_G) \tag{3.1}$$

where

$$I(e,E) = \begin{cases} 0 & \text{if } e \in E \\ 1 & \text{otherwise} \end{cases}$$

and $S = (V_S, E_S)$ is an induced graph $H[V_S]$, $V_S = \{g(v) | v \in V_G\}$.

## 3.3 Multi-objective Fitness Function

In designing heuristics or evolutionary algorithms for an optimization problem, we have to choose an appropriate fitness function which leads to an effective search path. In this respect, the fitness function $f_1$ described in Equation 3.1 is not an ideal fitness function, because it does not reflect the property of the subgraph isomorphism as follows:

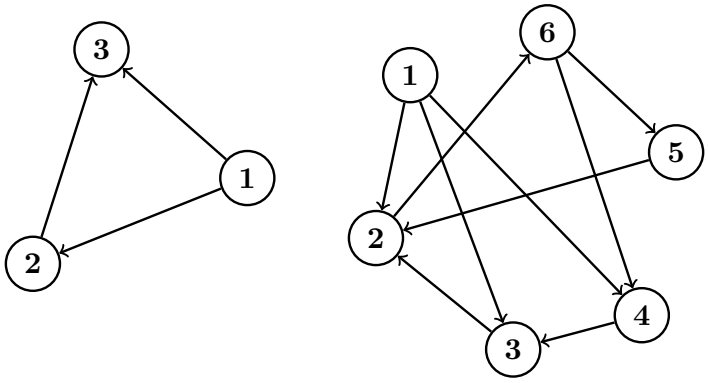
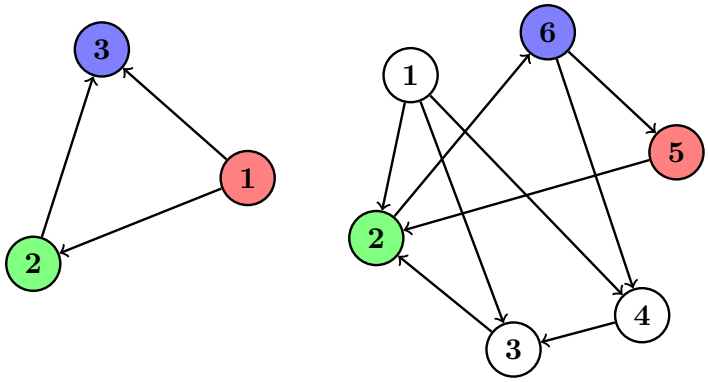


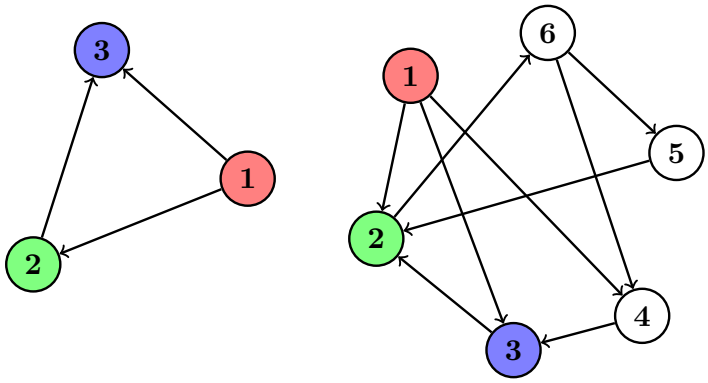

Figure 2: Two graph instances $G$ and $H$

**Corollary 1.** *Let $d^-(v)$ be a incoming degree and $d^+(v)$ be a outgoing degree of a vertex v. Given two directed graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, a vertex $u \in V_G$ cannot mapped by a subgraph isomorphism to vertex $v \in V_H$ unless $d^-(u) \leq d^-(v)$ and $d^+(u) \leq d^+(v)$, for $\forall u \in V_G$ and $\forall v \in V_H$.*

This problem was first discovered in [CYM12] and we describe the details with a example of Figure 2 and Figure 3. Figure 2 shows two graphs $G$ and $H$ and Figure 3 shows two different mappings, $g_1$ (Figure 3(a)) and $g_2$ (Figure 3(b)), between $G$ and $H$. The fitness value $f_1(g_1)$ and $f_1(g_2)$, a number of mismatched edges between $G$ and $H$, are two in both cases. However, $g_2$, in fact, is a better solution than $g_1$. In Figure 2, two vertices out of three are mapped to unsuitable vertices. Vertex 5 of $H$ has only one outgoing edge, while Vertex 1 of $G$ has two outgoing edges. Vertex 6 of $H$ has one incoming edge, while Vertex 3 of $G$ has two incoming edges. As shown in Corollary 1, a vertex of graph $G$ cannot be mapped to a vertex of graph $H$ if a number of incoming or outgoing edges of a vertex of $G$ is




(a) an mapping $g_1$




(b) an mapping $g_2$

Figure 3: Two possible mappings between $G$ and $H$

larger than those of a vertex of $H$. It is recommended that Vertex 1 of $G$ is not mapping to Vertex 5 of $H$. This is the same in case of mapping Vertex 3 in $G$ and Vertex 6 in $H$.

On the other hand, in case of $g_2$, there are no violation in respect of degree. The incoming and outgoing degrees of vertices 1, 2 and 3 of $G$ are smaller or equal than those of the vertex 1, 2 and 3 of $H$, respectively. It can become an optimal solution by only exchange a mapping of the vertex 2 in

$G$ to the vertex 4 in $H$.

Therefore, the degree constraint of a mapping $g : V_G \to V_H$ is formulated with following function:

$$f_2(g) = \sum_{v \in V_G} J(v), \tag{3.2}$$

where

$$J(v) = \begin{cases} 0 & \text{if } d^-(v) \leq d^-(g(v)) \text{ and } d^+(v) \leq d^+(g(v)) \\ 1 & \text{otherwise.} \end{cases}$$

The function $f_2(g)$ has no meaning by itself and needs to be combined with the function $f_1(g)$. It is clear that both of the value of $f_1(g)$ and the value of $f_2(g)$ equal zero when a mapping become a subgraph isomorphism. If the value of $f_1(g)$ equals zero, then it indicates an optimal solution. It is a sufficient condition for an optimal solution. However, the value of $f_2(g)$ begin zero is a necessary condition. Therefore, we combine both function linearly make a multi-objective fitness function as follows:

$$f(g,w) = w \cdot f_1(g) + (1-w) \cdot f_2(g). \tag{3.3}$$

Still, the goal is to minimize the value of $f_1(g)$. Since $f(w,g) = 0$ means $f_1(g) = 0$, optimizing $f(w,g)$ results in optimizing $f_1(g)$. For example, in

case of $w = 0.5$, the fitness values of $g_1$ and $g_2$ in Figure 3 are

$$\begin{aligned} f(0.5, g_1) &= 0.5 \cdot 2 + 0.5 \cdot 2 = 2 \quad \text{and} \\ f(0.5, g_2) &= 0.5 \cdot 2 + 0.5 \cdot 0 = 1. \end{aligned}$$

## 3.4 Local Heuristics

Kim and Moon [KM10] first used a simple two-vertex exchange heuristic for the subgraph isomorphism problem. The two-vertex exchange heuristic is similar to 2-OPT in the Traveling salesman problem. After then, Choi et. al [CKM14] refined this heuristic by swapping a mapping of all possible pairs of vertices until there is no further improvement.

The details are described in Algorithm 2.

**Algorithm 2:** Vertex swap local optimization algorithm

**Input**: A chromosome $C$ of **SIP**$(G,H)$

1: $L \leftarrow \{(i,j) \mid 1 \le i \le |V_G|,\ i < j \le |V_H|\}$
2: **repeat**
3: $flag \leftarrow false$
4: **for all** $(i,j) \in L$ in random order **do**
5: $swap\ (C[i], C[j])$
6: calculate the difference
7: **if** improved **then**
8: $flag \leftarrow true$
9: **else**
10: $swap\,(C[i], C[j])$ // cancel
11: **end if**
12: **end for**
13: **until** flag

This heuristic works differently depending on the fitness function, which

determines the direction of the search path of the heuristics. If the fitness function is able to reflect more characteristics of excellent solutions in relation to the current solution, the heuristic may take more efficient path to find the optimal solution. Since a local search algorithm should terminate in a reasonable number of steps, it usually make a sequence of greedy choices based on the state of a solution. It moves from one solution to another according to certain criteria. This algorithm compares the fitness value of the current solution with the fitness value of an adjacent solution. It does not take a path which makes the fitness function value lower, even though taking the path eventually leads to an optimal solution. So if we can design a fitness function such that a solution with the potential is rated high, it will help in taking a better search path.

## 3.5 Experiments

### 3.5.1 Experimental Setting

We generated random graphs by following a widely-used graph generation process for the subgraph isomorphism problem [CYM12, FSV01, CFV07].

First, a graph $H$ is generated by randomly selecting $\eta|V_H|^2$ directed edges among $|V_H|$ vertices without any other constraint, where $\eta$ denotes the edge density of a graph $H$. And then, a smaller graph $G$ is generated by sampling $|V_G|$ vertices from $H$ and selecting the induced subgraph $H[V_G]$. This means that there is always a subgraph isomorphism from $G$ to $H$ and the optimal fitness function value is always zero.

We selected 20 classes using 4 kinds of $\eta$ values of $H$ and 5 kinds of $|V_G|$ values, and independently generated 10 pairs of graph instances for each class, so that a total 200 pairs of graph instances were used for our experiments. We chose 0.01, 0.05, 0.1 and 0.5 for $\eta$ and 10, 30, 50, 70 and 90 for $|V_G|$. The number of vertices of the larger graph, $|V_H|$, was fixed to 100.

All algorithms implemented with in C++ language, compiled by g++ 4.8.4 with -O3 option and executed with Intel Xeon CPU E5-2660 v3 @ 2.60GHz and 1GB memory.

### 3.5.2 Comparison of Single and Multi-objective Function

We firstly compared the performance of three different fitness functions

$$\begin{aligned} f(1,g) &= f_1(g), \\ f(0.5,g) &= 0.5 \cdot f_1(g) + 0.5 \cdot f_2(g), \quad \text{and} \\ f(0.1,g) &= 0.1 \cdot f_1(g) + 0.9 \cdot f_2(g) \end{aligned}$$

in Algorithm 2. We conducted 100 runs for each instance to test and averaged the results of all instances, 1,000 runs, in each class.

Table 3.5.2 shows the average values of $f_1(g)$ and $f_2(g)$ before and after running Algorithm 2 for each fitness function. Based on the function $f_1$, the results of applying a multi-objective function showed better in 19 classes out of 20 classes, and specially $f(0.1,g)$ showed best results in 13 classes. Moreover, based on the function $f_2$, the results of three functions

Table 1: The comparison of three different fitness functions

| $\|V_1\|$ | $\eta$ | Before | | $f(1.0,g)$ | | $f(0.5,g)$ | | $f(0.1,g)$ | |
|---|---|---|---|---|---|---|---|---|---|
| | | $f_1(g)$ | $f_2(g)$ | $f_1(g)$ | $f_2(g)$ | $f_1(g)$ | $f_2(g)$ | $f_1(g)$ | $f_2(g)$ |
| 10 | 0.01 | 4.25 | 2.96 | 1.21 | 1.54 | 0.41 | 0.03 | **0.36** | **0** |
| | 0.05 | 13.11 | 0.71 | **4.43** | **0.35** | 4.45 | 0.06 | 4.65 | 0 |
| | 0.1 | 15.83 | 0.06 | 4.82 | 0.05 | **4.61** | **0.01** | 4.72 | 0 |
| | 0.2 | 23.01 | 0 | 6.93 | 0 | **6.86** | **0** | 6.87 | 0 |
| 30 | 0.01 | 18.61 | 11.84 | 6.91 | 7.25 | 5.33 | 0.56 | **4.96** | **0** |
| | 0.05 | 63.7 | 5.88 | 33.24 | 3.38 | 32.57 | 0.53 | **31.92** | **0** |
| | 0.1 | 105.7 | 1.71 | 57.38 | 0.53 | **57.06** | **0.13** | 57.21 | 0 |
| | 0.2 | 164.98 | 0.16 | 88.78 | 0.02 | **88.46** | **0.02** | 88.71 | 0 |
| 50 | 0.01 | 36.13 | 22.66 | 16.5 | 15.97 | 13.78 | 2.14 | **13.63** | **0.17** |
| | 0.05 | 146.89 | 16.59 | 88.68 | 11.05 | 88.11 | 2.69 | **87.69** | **0** |
| | 0.1 | 256.7 | 9.39 | 162.32 | 5.17 | **160.26** | **1.35** | 162.38 | 0 |
| | 0.2 | 428.13 | 2.27 | 276.89 | 0.9 | **275.21** | **0.22** | 276.06 | 0 |
| 70 | 0.01 | 59.81 | 36 | 31.35 | 28.37 | 29.01 | 6.7 | **28.1** | **1.89** |
| | 0.05 | 271.18 | 36.7 | 185.39 | 29.88 | 184.44 | 12.16 | **145.04** | **0.88** |
| | 0.1 | 476.02 | 28.87 | 334.52 | 20.35 | 335.35 | 10.39 | **292.96** | **0.15** |
| | 0.2 | 815.94 | 17.74 | 585.19 | 10.19 | 574.08 | 5.26 | **571.52** | **0.02** |
| 90 | 0.01 | 84.81 | 49.17 | 48.66 | 42.3 | 47.33 | 13.74 | **45.62** | **5.96** |
| | 0.05 | 404.36 | 56.78 | 293.29 | 52.42 | 269.53 | 29.84 | **24.68** | **0.86** |
| | 0.1 | 741.88 | 54.16 | 561.1 | 49.89 | 527.11 | 33.18 | **28.73** | **0.41** |
| | 0.2 | 1312.29 | 51.12 | 1006.17 | 45.45 | 946.02 | 33.76 | **146.2** | **0.81** |

were significantly different.

Figure 4 plots the values of three fitness functions which change with the number of swap operations in Algorithm 2 for an random instance of class $|V_1| = 50$ and $\eta = 0.01$. Comparing the final value of the $f_1(g)$ in the three cases, the $f(0.1,g)$ built the best solution. In Figure 4(a), the value of $f_1(g)$ declines over step, but the value of $f_2(g)$ does not decrease enough. It means that the output was far from satisfying the necessary condition to be an optimal solution. On the other hand, in case of Figure 4(b) and Figure 4(c), both value are deceasing rapidly. This confirms that the new

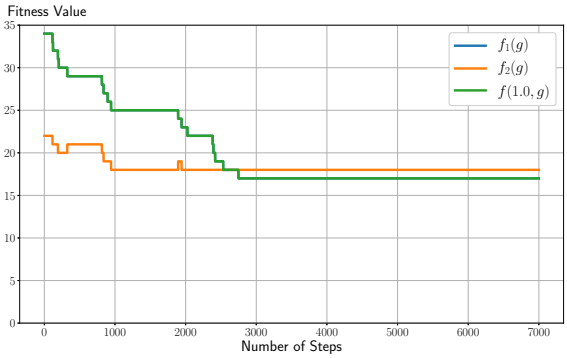
function derived by the degree constraints made the synergetic power with the main fitness function. Therefore, it can be concluded that combining new function based on degree constraint facilitates optimizing of the original fitness function.

### 3.5.3 Global Convexity of the Multi-objective Fitness Landscape

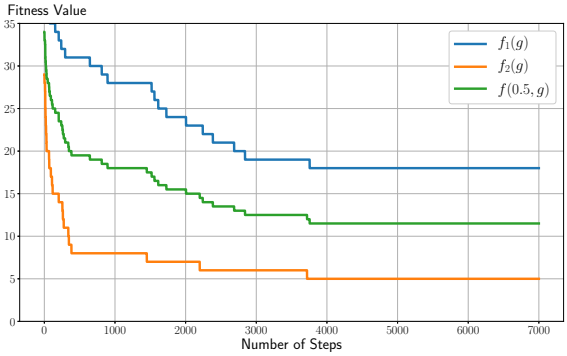
Given a set of local optima, Boese et al. [BKM94] plotted, for each local optimum, the relationship between the cost and the average distance from all the other local optima. They conducted experiments for the graph bisection problem and the traveling salesman problem, and found strong positive correlation in both problems. This fact hints that the best local optimum is highly probably located near the center of the local optimum space and, roughly speaking, the local optimum space is globally convex.

We applied their approach to the subgraph isomorphism problem. From 2,500 randomly generated solutions, the local minima were obtained by applying Algorithm 2 and the average Hamming distances between one local minimum to all other local minimum were calculated. After then, we calculated the correlation coefficient between the fitness value and the average Hamming distance.

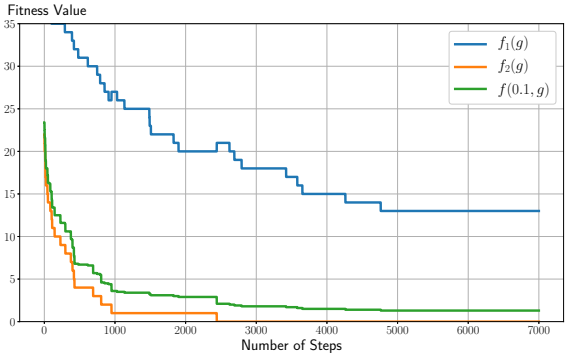
Table 2 represents the correlation coefficients for each class and fitness function. Figure 5 denotes a sample plotting of an instance of $|V_1| = 70$ and $\eta = 0.05$ class. For all classes except class $(10, 0.01)$ and $(10, 0.1)$, the results of multi-objective functions $f(0.5, g)$ and $f(0.1, g)$ showed stronger




(a) $f(1.0, g)$




(b) $f(0.5, g)$




(c) $f(0.1, g)$

Figure 4: Local optimization paths with three different fitness functions

Table 2: Fitness value-average distance correlation coefficients

| $\lvert V_1 \rvert$ | $\eta$ | $f(1.0,g)$ | $f(0.5,g)$ | $f(0.1,g)$ |
|---|---|---|---|---|
| 10 | 0.01 | **0.219** | 0.128 | 0.193 |
| | 0.05 | 0.106 | 0.092 | **0.287** |
| | 0.1 | **0.486** | 0.465 | 0.457 |
| | 0.2 | 0.238 | 0.187 | **0.257** |
| 30 | 0.01 | 0.255 | **0.380** | 0.3547 |
| | 0.05 | 0.059 | 0.153 | **0.302** |
| | 0.1 | 0.121 | 0.139 | **0.161** |
| | 0.2 | 0.149 | 0.154 | **0.207** |
| 50 | 0.01 | 0.228 | **0.403** | 0.3405 |
| | 0.05 | 0.045 | 0.151 | **0.226** |
| | 0.1 | 0.071 | **0.207** | 0.157 |
| | 0.2 | 0.115 | 0.144 | **0.213** |
| 70 | 0.01 | 0.221 | **0.555** | 0.4682 |
| | 0.05 | 0.060 | 0.300 | **0.631** |
| | 0.1 | 0.192 | 0.474 | **0.482** |
| | 0.2 | 0.294 | **0.325** | 0.303 |
| 90 | 0.01 | 0.263 | 0.602 | **0.631** |
| | 0.05 | 0.331 | 0.637 | **0.766** |
| | 0.1 | 0.401 | **0.737** | 0.658 |
| | 0.2 | 0.437 | 0.581 | **0.611** |

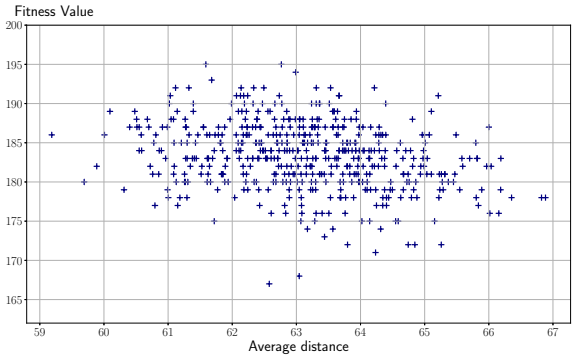
positive fitness-distance correlation than those of $f(1.0, g)$. Instances of class $V_1 = 10$ are too small for fitness functions to affect the structure of problem space.

From the experiment results, we infer that the fitness landscape of multi-objective fitness function is more globally convex than that with single-objective fitness function for sufficiently large size instance of the subgraph isomorphism problem.

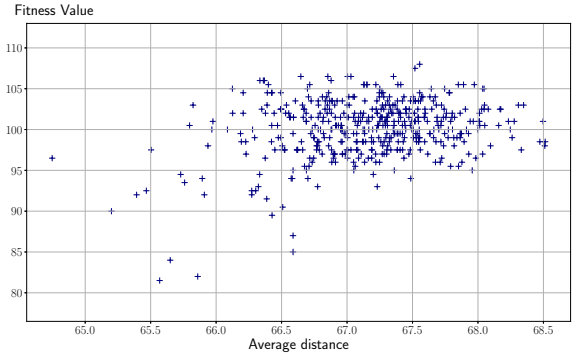
### 3.5.4 Hybrid Genetic Algorithm

We designed a hybrid genetic algorithm ($GA(0.1, 0.9)$) for the subgraph isomorphism problem with a multi-objective fitness function and compared the performance with the previously genetic algorithm ($GA(1.0, 0.0)$) proposed by Kim et. al [KM10]. We measured performance and execution time according to various $w$ values through experiments. Figure 6 shows the average fitness values and the running times as the value of $w$ changes. Except for the case of $w = 1.0$ used in [KM10], it showed a fairly uniform performance for the other values. We chose $w = 0.1$, which was the most advantageous in terms of the running time.

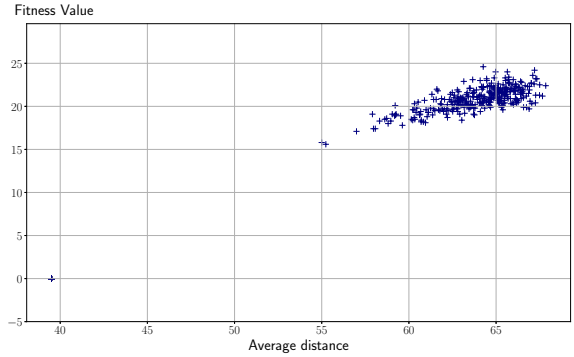
The experimental setting of both algorithm were same and only the fitness function was set differently: $GA(1.0, 0.0)$ used the single objective function $f(1.0, g)$ and $GA(0.1, 0.9)$ used the multi-objective fitness function $f(0.1, g)$. For each instance, we conducted 1000 runs for each instance to test the algorithms and averaged the results of all instances in each class and measured the average fitness value, the average running time, and the ratio of finding optimal solutions.




(a) $f(1.0, g)$




(b) $f(0.5, g)$




(c) $f(0.1, g)$

Figure 5: Plots of fitness value versus average distance among local optima

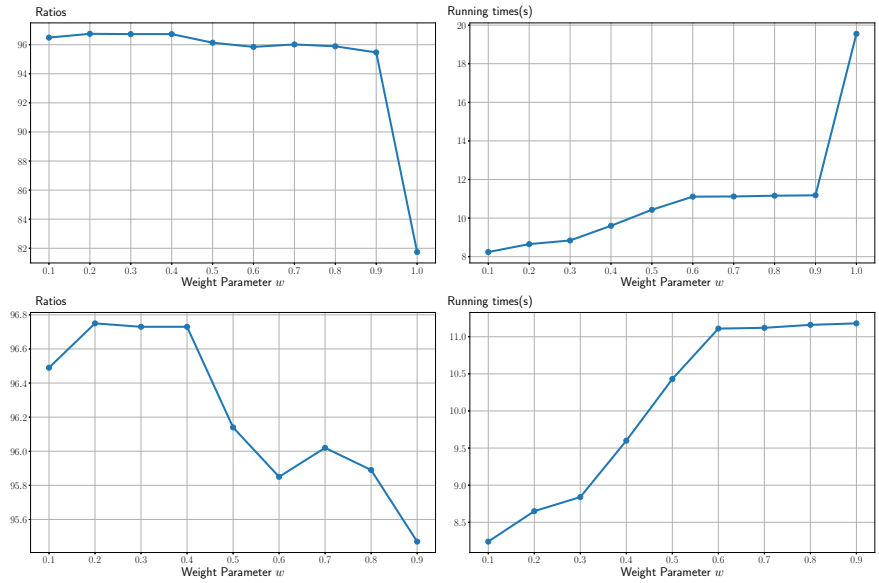



Figure 6: Comparative analysis of the fitness function $f$ for the fitness value and the running time with different weight parameter $w$

Table 3 shows the result of both algorithm. In the ten classes, $GA(0.1, 0.9)$ showed the better performance than $GA(1.0, 0.9)$, marked in bold. Above all, it is noticeable that there were dramatic performance improvement for the three classes $(50, 0.01)$, $(70, 0.01)$, and $(90, 0.01)$, in which the previous algorithms did not perform well. The overall ratios finding optimal solutions were also improved from 81.78% to 96.56% for all classes. Through the experimental results, we conclude that the well-designed fitness function is helpful to improve the performance of the hybrid genetic algorithm.

Table 3: Overall results of the hybrid GA compared to the previous work. The results which are better than the previous work are shown in bold.

| $\|V_G\|$ | $\eta$ | $GA(1.0, 0.0)$ | | | | $GA(0.1, 0.9)$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $f$ average | $f$ SD | Time | Ratio | $f$ average | $f$ SD | Time | Ratio |
| 10 | 0.01 | 0.0000 | 0.0000 | 0.03 | 100.00% | 0.0000 | 0.0000 | 0.03 | 100.00% |
| | 0.05 | 0.0020 | 0.0447 | 0.05 | 99.80% | **0.0002** | **0.0042** | **0.06** | **99.82%** |
| | 0.1 | 0.0023 | 0.0479 | 0.08 | 99.77% | 0.0002 | 0.0048 | 0.11 | 99.77% |
| | 0.2 | 0.4256 | 0.9002 | 0.55 | 79.17% | 0.0427 | 0.0895 | 0.93 | 78.85% |
| 30 | 0.01 | 0.0059 | 0.0766 | 1.13 | 99.41% | **0.0000** | **0.0000** | **0.39** | **100.00%** |
| | 0.05 | 5.0088 | 7.0046 | 6.60 | 60.96% | **0.2412** | **0.5020** | **7.09** | **78.73%** |
| | 0.1 | 5.9865 | 16.2949 | 4.16 | 87.99% | **0.4723** | **1.4629** | **6.31** | **90.43%** |
| | 0.2 | 0.0226 | 1.5980 | 1.04 | 99.98% | 0.0047 | 0.2350 | 2.52 | 99.96% |
| 50 | 0.01 | 2.2322 | 1.3002 | 15.19 | 4.80% | **0.0043** | **0.0203** | **7.29** | **95.69%** |
| | 0.05 | 8.6474 | 25.8336 | 10.50 | 89.78% | **0.0015** | **0.1093** | **3.51** | **99.98%** |
| | 0.1 | 0.0875 | 4.3795 | 3.72 | 99.96% | **0.0000** | **0.0000** | **3.65** | **100.00%** |
| | 0.2 | 0.0000 | 0.0000 | 1.51 | 100.00% | 0.0000 | 0.0000 | 3.86 | 100.00% |
| 70 | 0.01 | 16.0792 | 5.2860 | 32.32 | 0.00% | **0.0126** | **0.0351** | **14.48** | **88.01%** |
| | 0.05 | 0.9251 | 14.4306 | 9.41 | 99.59% | **0.0000** | **0.0000** | **2.76** | **100.00%** |
| | 0.1 | 0.0000 | 0.0000 | 3.49 | 100.00% | 0.0000 | 0.0000 | 3.92 | 100.00% |
| | 0.2 | 0.0000 | 0.0000 | 2.37 | 100.00% | 0.0000 | 0.0000 | 6.52 | 100.00% |
| 90 | 0.01 | 8.9344 | 8.0415 | 52.60 | 14.52% | **0.0000** | **0.0000** | **6.75** | **100.00%** |
| | 0.05 | 0.0000 | 0.0000 | 4.93 | 100.00% | 0.0000 | 0.0000 | 3.52 | 100.00% |
| | 0.1 | 0.0000 | 0.0000 | 3.63 | 100.00% | 0.0000 | 0.0000 | 4.55 | 100.00% |
| | 0.2 | 0.0000 | 0.0000 | 3.43 | 100.00% | 0.0000 | 0.0000 | 7.67 | 100.00% |

# Chapter 4

# Incremental Hybrid Genetic Algorithm

## 4.1 Incremental Process

The incremental genetic algorithm (IGA) is a method of dividing a problem into successive subproblems and solving them sequentially to obtain solutions to the original problem [CKM14].

In this process, each subproblem is solved by a genetic algorithm and the obtained solutions are extended as the initial solutions of the next subproblem. We explain in detail how this algorithm works for the subgraph isomorphism problem.

Let a substructure of a graph $G = (V, E)$ be a subset of $V$ and their connected edges in the graph $G$. In this problem, it means a subgraph. On the same problem structure, if the input is changed from the original graph to its substructure, we call it a subproblem. For example, in the subgraph isomorphism problem, $\mathbf{SIP}(G', H)$ is a subproblem of $\mathbf{SIP}(G, H)$ where $G' \subseteq G$.

Consider a finite subsequence of consecutive subproblems $\{\mathbf{SIP}(G_1, H)$, $\mathbf{SIP}(G_2, H), \ldots, \mathbf{SIP}(G_n, H)\}$, where $G_i \subseteq G$ for $1 \leq i \leq n$. Let $G_1$ be a sufficiently small subgraph of $G$ and $G_n$ be the same graph as $G$. In this case, if we add extra mappings to the solutions obtained after solving the sub-

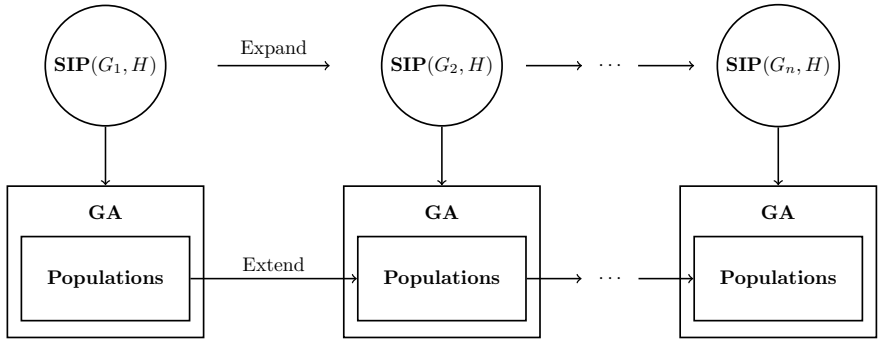



Figure 7: Overview of an incremental genetic algorithm

problem **SIP**$(G_i, H)$, these extended solutions are likely to be good quality initial solutions for the next subproblem **SIP**$(G_{i+1}, H)$. If we repeat this task sequentially, we will eventually get the solutions of the original problem. Figure 7 shows an overview of an incremental genetic algorithm.

The rationale behind the IGA is that high quality solutions of the one subproblem probably provide good initial points of the next subproblem [CKM14, BBK11, MAEF06, VN10, WYJ$^+$04]. If the subproblems in the sequence are ordered by the graph size of the subproblems, the optimal solutions of the subproblem with small size graph can be found easily and these are expected to be extended as good solutions for the next subproblem. Moreover, in the subgraph isomorphism problem, every sequence of the subproblems has the optimal substructure because the subgraph relation is transitive, i.e., if $G_1$ is a subgraph of $G_2$ and $G_2$ is a subgraph of $G_3$, then $G_1$ is a subgraph of $G_3$. This property makes the application of an incremental approach to the subgraph isomorphism problem more appropriate than other combinatorial optimization problems such as MAX-CUT or TSP.

To precisely illustrate these aspects, we say that a sequence of subprob-

lems has an optimal substructure, if one of the optimal solutions of each subproblem could be extended to the optimal solution of the next subproblem. Existence of an optimal substructure conceptually explains that we are able to solve the original problem by enumerate all of the optimal solutions for each of the subproblems. Of course, the presence of an optimal structure does not guarantee that the IGA can always find optimal solutions. Not every sequence having an optimal substructure leads to a good solution. If there exist a large number of optimal solutions for intermediate subproblems, and only a few of them are extendable to optimal solutions for the original problem, then the IGA is less likely to find promising solutions.

The key point of the IGA is how to design a sequence of subproblems. First, the order in which vertices are added to expand a graph, determines the overall search path, which has already been identified in previous backtracking methods. Second, the expansion size between the subproblems, i.e. a number of vertices to expand a graph from one subproblem to the next, determines the size of the problem space covered by GA. If we set the expansion size to be small, GA will be able to exploit the problem space in detail for each problem, but it takes a lot of time and is inefficient in terms of overall execution time. On the other hand, too large expansion size makes it difficult for GA to evolve previous solutions into good solutions. Therefore, it is necessary to set the appropriate expansion size and the order of subproblems considering the search capability of GA for each subproblem.

## 4.2 Proposed Algorithm

We present our incremental hybrid genetic algorithm (IHGA) for the subgraph isomorphism problem in Algorithm 3.

**Algorithm 3:** Incremental process for the subgraph isomorphism problem

**Input**: $G = (V_G, E_G)$, $H = (V_H, E_H)$
**Output**: A injective function $g : V_G \to V_H$

1: $V' \leftarrow$ Reordering($V_G$)
2: $n \leftarrow$ the number of subproblems
3: $\mathbf{m} = \{m_1, m_2, \ldots, m_n\} \leftarrow$ the sequence of expansion size
4:
5: $G_0 \leftarrow \varnothing$
6: $P_0 \leftarrow$ random initial population of **SIP**$(G_0, H)$
7: **for** $i = 1$ **to** $n$ **do**
8: $\quad V_{curr} \leftarrow \{V'_1, \ldots, V'_{m_i}\}$
9: $\quad V' \leftarrow V' - V_{curr}$
10: $\quad G_i \leftarrow$ `induced subgraph` $G[V_{G_{i-1}} \cup V_{curr}]$
11: $\quad P_i \leftarrow$ initial population generated by $P_{i-1}$
12: $\quad P_i \leftarrow$ hybrid GA($P_i$)
13: **end for**
14: **return** the best in $P_n$

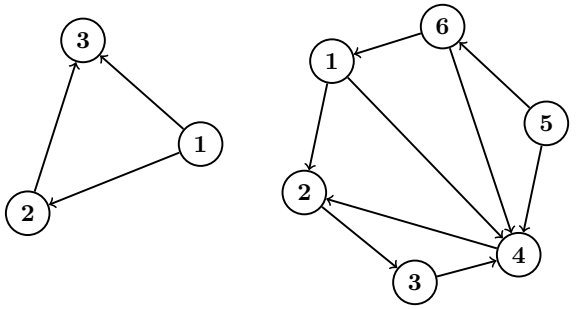
The first half of the algorithm sets up the incremental process by decomposing the original problem into the sequence of consecutive subproblems. The sequence is determined by the number of subproblems, $n$, the number of vertices to be added at each subproblem, $\mathbf{m}$, and the order of vertices, $V'$.

In line 1, the order of vertices added is determined by the vertex reordering scheme. According to previous study [CKM14], the vertex join order is one of the most important factors for the performance of an incremental genetic algorithm. The reordering schemes we used are described in

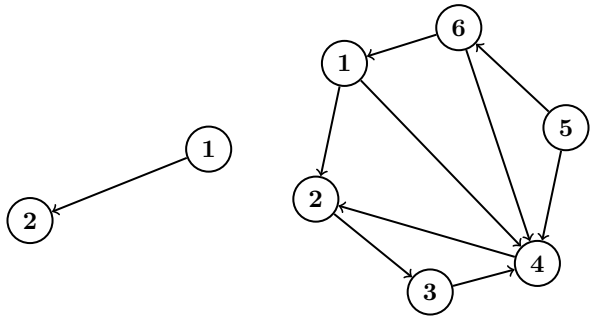
detail in the next section. The number of subproblems and the expansion size of each subproblem are determined in lines 2 and 3, which decide the problem space size of each subproblem. If we improve the performance of hybrid genetic algorithm, we can increase the size of problem space of each subproblem.

In the second half of the algorithm, it is shown how to solve the subproblems and extending solutions to obtain the original solution problems. We starts from the problem $\mathbf{SIP}(G_0, H)$ with an empty domain graph, $G_0 = \varnothing$. For every $i^{\text{th}}$ subproblem, we expand $G_{i-1}$ into $G_i$ by adding $m_i$ vertices and edges between $V_{G_{i-1}}$ and new added vertices. And then, hybrid genetic algorithm takes the results of the previous subproblem to build an initial population of $\mathbf{SIP}(G_i, H)$ and evolves it over generations. The solution of the original problem $\mathbf{SIP}(G, H)$ is obtained from the results of the $\mathbf{SIP}(G_n, H)$.

Figure 8 shows how the incremental process works. Consider two graphs in Figure 8(a). In this case, the number of subproblems is 3 and the expansion size for each subproblem, $m_i$, is 1. Figure 8(b) describes the second subproblem $\mathbf{SIP}(G_2, H)$ and the population $P_2$ evolved by hybrid GA. From the second subproblem $\mathbf{SIP}(G_2, H)$ and the population $P_2$, Figure 8(c) shows how to obtain the graph $G_3$ from $G_2$ and the population $P_3$ from $P_2$ for initializing the third subproblem $\mathbf{SIP}(G_3, H)$. Graph $G_2$ is expanded into $G_3$ by adding one vertex 3 and two edges $(1,3)$ and $(2,3)$. And the population $P_3$ is initialized from $P_2$ by adding extra mappings for a new added vertex 3, which are denoted in bold.

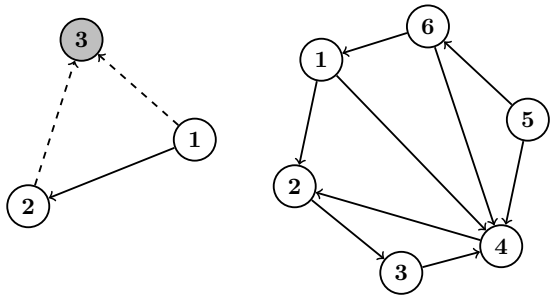



(a) Two graphs $G$ and $H$




| $P_2^1$ | $P_2^2$ | $P_2^3$ | $\cdots$ |
|---|---|---|---|
| $1 \to 2$ | $1 \to 6$ | $1 \to 4$ | |
| $2 \to 3$ | $2 \to 1$ | $2 \to 2$ | $\cdots$ |

(b) Population $P_2$ evolved in subproblem $\mathbf{SIP}(G_2, H)$




| $P_3^1$ | $P_3^2$ | $P_3^3$ | $\cdots$ |
|---|---|---|---|
| $1 \to 2$ | $1 \to 6$ | $1 \to 4$ | |
| $2 \to 3$ | $2 \to 1$ | $2 \to 2$ | $\cdots$ |
| $\mathbf{3 \to 4}$ | $\mathbf{3 \to 5}$ | $\mathbf{3 \to 3}$ | |

(c) Initial population $P_3$ of $\mathbf{SIP}(G_3, H)$ extended from $P_2$

Figure 8: Description of expanding graph $G_i$ and initializing population $P_i$ in the incremental algorithm

## 4.3 Design Schemes

### 4.3.1 Vertex Reordering

The vertex join order is one of the most important design issues to determine the performance of an incremental genetic algorithm. Well-designed ordering scheme can prune out infeasible problem space early in the search process and allow to select an efficient overall search path. Information on vertex adjacency and the degree of vertices has been used in the ordering method of graph based problem [LHKL12, HKM06] and we applied three reordering schemes for the vertices of $G$ from [CKM14] and considered an ordering newly.

- Max-degree ordering (MD)

  We sort the vertices in non-increasing order of degree. The degree of a vertex is the sum of both ingoing and outgoing degrees.

- BFS ordering (BFS)

  We randomly select a starting vertex, and then run the breadth-first search on $G$. When the graph is disconnected and not all of the vertices are visited, I randomly choose another unvisited vertex and continue the traverse.

- Max-adjacency ordering (MA)

  We randomly select a starting vertex, and repeatedly select one of the most attractive vertex in a greedy manner. The attractiveness of a vertex $v$ is the number of adjacent vertices that are already ordered. Two

vertices are adjacent to each other if there is an edge in any direction.

- Max-degree-adjacency ordering (MDA)

  We add a tie-breaking rule to Max-degree ordering; in a case of a tie, the vertex having more adjacency to vertices in previous subproblem comes earlier in the ordering. This is a combination of the Max-degree ordering and Max-adjacency ordering for a synergy effect.

### 4.3.2 Stopping Criterion

Basically, we use a fixed number of generations for all of the subproblems. But this may lead to an excessive number of generations in earlier subproblems, because hybrid GA may converge very fast for relatively small problems. Moreover, keeping some solutions that are not converged in the population may preserve the diversity of solutions. Both the quality and the diversity of population in the one subproblem can have a decisive effect on the next subproblem. So, if a certain percentage of the population for the one subproblem converges to the optimal solution, we terminate GA and move on the next subproblem. Initially, we regard $|V_H|$ generations as the unit of time and completely distributed a fixed number of generations equally to each subproblem. Before starting each subproblem, we redistribute the remaining generations equally to the remaining subproblems. By this procedure, we expect that more generations are assigned to the later larger subproblems to be evolved longer.

### 4.3.3 Expansion Size

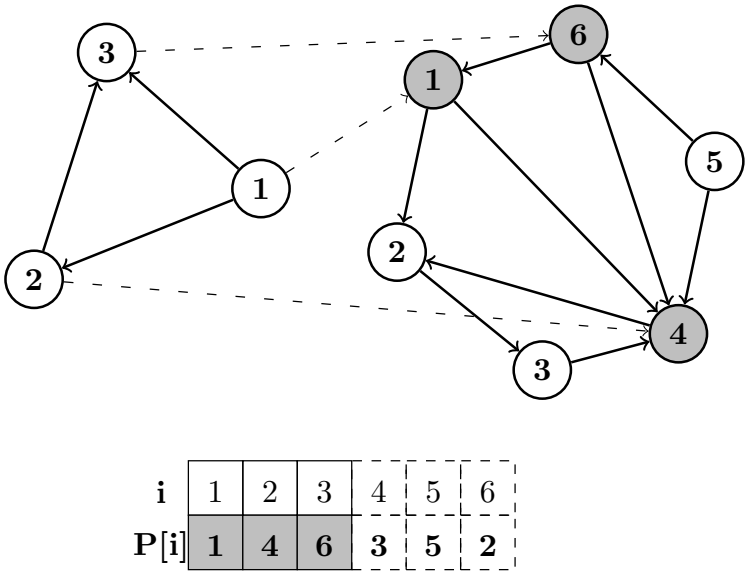
The number of vertices to be added also determines the problem space of each subproblem. A naive method is to add a single vertex at each step. But it will be a waste of time to run hybrid GA when the graph expanded is too simple. Adding more vertices at a step enables efficient space search, but an immoderate expansion size may cause the problem space to be too large for hybrid GA to cover. It is required to strike a balance between efficiency and difficulty by selecting an appropriate expansion size.

## 4.4 Genetic Frameworks

The hybrid genetic algorithm we used in the incremental process for the subgraph isomorphism problem is described below.

- Representation

  Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ where $|V_G| \leq |V_H|$, a chromosome represents a permutation of $V_H$ as an integer array. A mapping $g : V_G \rightarrow V_H$, a solution of **SIP**($V_G$, $V_H$), is decoded by first $|V_G|$ genes in the chromosome. A vertex $v_{G,i} \in V_G$ is mapped to $v_{H,p[i]} \in V_H$ and an edge $(v_{G,i}, v_{G,j}) \in E_G$ is mapped to $(v_{H,p[i]}, v_{H,p[j]}) \in E_H$. Figure 9 shows an example. The main advantage of this representation is the flexibility toward the problem size expansion. Since a chromosome already has a full permutation of $V_H$, I can easily crossover and mutate without extend the mapping at each subproblem without modifying values of the genes.




| i | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| P[i] | 1 | 4 | 6 | 3 | 5 | 2 |

Figure 9: Representation of a chromosome. Each vertex $i$ in $G$ is mapped by a vertex $P[i]$ in $H$, drawn by dashed line.

- Fitness function

  We use the multi-objective fitness function $f(w,g)$ introduced in Section 3.3. The parameter $w$ is set to $10/|V_H|$.

- Population management

  For the synthetic dataset, the population size of each subproblem in the incremental process is fixed to 100. The only initial population for the first subproblem is randomly generated. From the second subproblem, GA takes the population evolved in the previous subproblem as initial population of the current subproblem. For the real world dataset, we have experimented with different population sizes to compare the performance of IHGA with changes in genetic parameters.

- Selection

  The tournament selection is used. We pick two chromosomes randomly and return better one with 80 percent of chance, otherwise return the worse one.

- Crossover and mutation

  We used cycle crossover [OSH87]. For the mutation, we select a number of genes to shuffle them in random order. Each of the gene independently has 40 percent of mutational chance.

- Local heuristics

  We used Algorithm 2, two vertex exchange local heuristic, introduced in Section 3.4.

- Replacement

  We generate half of the population size offspring per generation and choose best solutions as many as the population size.

- Stopping criterion

  The hybrid GA is terminated when a certain ratio of the solutions in the population becomes optimal solutions. We use the ratio values ($TH$) of 1%, 50%, and 100%. Regardless of this criterion, in the last step, in which, the subproblem is the same as the original one, the algorithm stops if it finds the optimal solution. If we set $TH$ as $\infty$, the hybrid GA unconditionally executes a fixed number of generations and ends.

## 4.5 Experimental Results

We had experiments with the proposed incremental hybrid genetic algorithm for two kinds of datasets. The effects of design schemes proposed in Section 4.3 were verified and our algorithm was compared with previous algorithms on synthetic data consisting of random graphs. We also tested our algorithm for the real world data with large size graphs. All algorithms implemented with in C++ language, compiled by g++ 4.8.4 with -O3 option and executed with Intel Xeon CPU E5-2660 v3 @ 2.60GHz and 1GB memory.

### 4.5.1 Synthetic Data

#### 4.5.1.1 Dataset and Evaluation

We generated random graphs by following a widely-used graph generation process for the subgraph isomorphism problem [CYM12, FSV01, CFV07], as same way as described in Section 3.5.1. First, a graph $H$ is generated by randomly selecting $\eta|V_H|^2$ directed edges among $|V_H|$ vertices without any other constraint, where $\eta$ denotes the edge density of a graph $H$. And then, a smaller graph $G$ is generated by sampling $|V_G|$ vertices from $H$ and selecting the induced subgraph $H[V_G]$. This means that there is always a subgraph isomorphism from $G$ to $H$ and the optimal fitness function value is always zero.

We selected 20 classes using 4 kinds of $\eta$ values of $H$ and 5 kinds of $|V_G|$ values, and independently generated 10 pairs of graph instances for each class, so that a total 200 pairs of graph instances were used for our

Table 4: Results of different vertex reordering schemes. The best result for each class is shown in bold.

| $\|V_G\|$ | $\eta$ | $f$ average | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | **Kim** [KM10] | **Choi** [CYM12] | RAND | BFS | MD | MA | MDA |
| 10 | 0.05 | 0.0002 | 0.0002 | 0.0026 | 0.0009 | **0.0001** | 0.0005 | **0.0000** |
| | 0.1 | 0.0023 | 0.0002 | 0.0042 | 0.0006 | **0.0002** | 0.0007 | **0.0002** |
| | 0.2 | 0.4256 | 0.0427 | 0.0962 | 0.0691 | **0.0256** | 0.0495 | **0.0229** |
| 30 | 0.05 | 5.0088 | 0.2412 | 0.8088 | **0.2106** | **0.0959** | 0.0903 | **0.0560** |
| | 0.1 | 5.9865 | 0.4723 | 1.1702 | 0.6226 | **0.2622** | **0.2151** | **0.1973** |
| | 0.2 | 0.0226 | 0.0047 | 0.1227 | 0.0973 | 0.0049 | 0.0224 | 0.0062 |
| 50 | 0.01 | 2.2322 | 0.0043 | 0.0899 | **0.0005** | **0.0001** | **0.0006** | **0.0000** |
| | 0.05 | 8.6474 | 0.0015 | 0.1758 | 0.0483 | 0.0209 | 0.0112 | **0.0000** |
| 70 | 0.01 | 16.0792 | 0.0126 | 0.0914 | **0.0069** | **0.0013** | **0.0068** | **0.0001** |

experiments. We chose 0.01, 0.05, 0.1, and 0.5 for $\eta$ and 10, 30, 50, 70 and 90 for $|V_G|$. The number of vertices of the larger graph, $|V_H|$, was fixed to 100.

We conducted 1,000 runs for each instance to test the algorithms and averaged the results of all instances in each class. We measured the average fitness value, the average running time, and the ratio of finding optimal solutions.

We compared the proposed incremental hybrid genetic algorithm (IHGA) with the previously proposed genetic algorithms, Kim et. al. [KM10] (**Kim**) and Choi et. al. [CYM12] (**Choi**). These studies were chosen because they tested graphs with a number of vertices close to 100.

#### 4.5.1.2 Effect of Vertex Reordering

Table 4 shows the average fitness value of previous works and the IHGA with five different reordering schemes. For each subproblem, the ex-

pansion size is set to 1, and the stopping criterion threshold $TH$ is set to $\infty$, which means the hybrid GA is set to run for a fixed number of generations. The best results are shown in bold. Adding the incremental process to the hybrid GA with randomized vertex reordering degraded the performance in all of the 9 classes. On the other hand, the other schemes using the information of the vertices show more improved results, and we have found that the vertex ordering, which determines the order of expanding the graph $G$, is very important in the incremental process. The best reordering schemes for the incremental approach is max-degree-adjacency (MDA), even showed better performance than the previous work [CKM14]. We therefore will fix the reordering scheme as MDA in the rest of our experiments for the incremental process.

#### 4.5.1.3 Effect of Partial Random Initialization

Through the experiment in Section 4.5.1.2, we could conclude that performance improvement can be achieved by only applying the basic incremental process with the reordering scheme.

In fact, in order to verify the effectiveness of the incremental process itself, we randomly initialized a certain ratio of solutions before running the hybrid GA in each subproblem. We fixed the schemes of incremental genetic algorithm as the MDA reordering, $\infty$ stopping criterion threshold value, and the expansion size value of 1 which shows the best performance in Table 4 and only changed the ratio to 0%, 10%, 20%, 30%, 40%, and 50% for every subproblem. Table 5 denotes the average fitness value of each partial randomization. For most of the classes, the random initialization degraded

Table 5: Results of different partially random initializations in each subproblem

| $\|V_G\|$ | $\eta$ | $f$ average | | | | | | Corr. |
|---|---|---|---|---|---|---|---|---|
| | | 0% | 10% | 20% | 30% | 40% | 50% | |
| 10 | 0.05 | 0.0000 | 0.0001 | 0.0003 | 0.0004 | 0.0005 | 0.0003 | 0.8448 |
| | 0.1 | 0.0002 | 0.0002 | 0.0002 | 0.0002 | 0.0002 | 0.0003 | 0.2015 |
| | 0.2 | 0.0229 | 0.0225 | 0.0242 | 0.0234 | 0.0259 | 0.0274 | 0.8956 |
| 30 | 0.05 | 0.0560 | 0.0892 | 0.1445 | 0.2586 | 0.4503 | 0.6462 | 0.9573 |
| | 0.1 | 0.1973 | 0.2417 | 0.3454 | 0.5938 | 0.8790 | 1.2312 | 0.9619 |
| | 0.2 | 0.0062 | 0.0050 | 0.0165 | 0.0280 | 0.0702 | 0.1766 | 0.8537 |
| 50 | 0.01 | 0.0000 | 0.0002 | 0.0026 | 0.0101 | 0.0202 | 0.0318 | 0.9421 |
| | 0.05 | 0.0000 | 0.0057 | 0.0113 | 0.0458 | 0.1774 | 0.3129 | 0.8898 |
| 70 | 0.01 | 0.0001 | 0.0151 | 0.0374 | 0.0647 | 0.0847 | 0.1023 | 0.9972 |

the quality of solutions. Through correlation coefficients between random initialization and the average fitness value, it can be shown that the quality of solutions is declined more if more solutions are randomly initialized in each subproblem. Therefore, it is useful to evolve and extend solutions gradually and we will extend and reuse all solutions of the one subproblem as the initial solutions for the next subproblem.

#### 4.5.1.4 Stopping Criterion

Table 6 denotes the average fitness value of the IHGA with three different stopping criteria for each subproblem. The stopping criteria were applied to the algorithm with random reordering scheme and MDA reordering scheme. For each reordering scheme, I also denoted the result of an algorithm when threshold value is $\infty$. We marked the best result in bold for each class.

In general, reducing the threshold value showed better performance

for both reordering schemes. In the case with random reordering scheme, the case of lowest threshold value gave the best performance in all classes. When MDA reordering scheme was applied, there was no significant difference in the average fitness value between the case of 1% threshold value and that of 50% threshold value, but the case of 1% threshold value executed slightly more rapidly than the case of 50% threshold value. Since reducing the threshold value increases the diversity of solutions, focusing on exploration in intermediate steps seems to be more helpful than focusing on exploitation. Than evolving from a population full of local optima, it was better to evolve from a diverse population where only one of the solutions is locally optimal.

Table 6: Results of different stopping criteria

(a) Random reordering

| $\|V_G\|$ | $\eta$ | $f$ average | | | | Running time (s) | | |
|---|---|---|---|---|---|---|---|---|
| | | $\infty$ | 100% | 50% | 1% | 100% | 50% | 1% |
| 10 | 0.05 | 0.0026 | 0.0003 | 0.0003 | **0.0001** | 0.20 | 0.19 | **0.16** |
| | 0.1 | 0.0042 | 0.0006 | 0.0004 | **0.0003** | 0.33 | 0.28 | **0.20** |
| | 0.2 | 0.0962 | 0.0652 | 0.0583 | **0.0478** | 1.18 | 1.16 | **0.90** |
| 30 | 0.05 | 0.8088 | 0.6528 | 0.6304 | **0.5350** | 6.40 | 6.45 | **6.35** |
| | 0.1 | 1.1702 | 1.0699 | 1.0533 | **0.9319** | 6.81 | 6.79 | **6.16** |
| | 0.2 | 0.1227 | 0.1335 | 0.1483 | **0.1034** | 6.33 | 6.26 | **6.00** |
| 50 | 0.01 | 0.0899 | 0.0464 | 0.0396 | **0.0189** | 12.89 | 13.21 | **11.26** |
| | 0.05 | 0.1758 | 0.0973 | 0.1027 | **0.0935** | 15.00 | 14.86 | **14.07** |
| 70 | 0.01 | 0.0914 | 0.0915 | 0.0915 | **0.0731** | 28.61 | 28.54 | **25.99** |

(b) MDA reordering

| $\|V_G\|$ | $\eta$ | $f$ average | | | | Running time (s) | | |
|---|---|---|---|---|---|---|---|---|
| | | $\infty$ | 100% | 50% | 1% | 100% | 50% | 1% |
| 10 | 0.05 | 0.0000 | **0.0000** | 0.0000 | 0.0001 | 0.17 | 0.16 | **0.14** |
| | 0.1 | 0.0002 | 0.0001 | **0.0000** | 0.0001 | 0.25 | 0.23 | **0.18** |
| | 0.2 | 0.0229 | 0.0157 | **0.0127** | 0.0137 | 0.69 | 0.70 | **0.40** |
| 30 | 0.05 | 0.0560 | 0.0406 | 0.0388 | **0.0347** | 3.96 | 3.74 | **3.36** |
| | 0.1 | 0.1973 | 0.1718 | 0.1695 | **0.1680** | 4.25 | 4.19 | **4.04** |
| | 0.2 | 0.0062 | 0.0061 | **0.0049** | 0.0088 | 4.63 | **4.58** | 4.72 |
| 50 | 0.01 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 9.89 | 9.04 | **7.71** |
| | 0.05 | 0.0000 | 0.0008 | 0.0008 | **0.0000** | 11.96 | 11.92 | **11.81** |
| 70 | 0.01 | 0.0001 | **0.0001** | 0.0001 | 0.0002 | 25.10 | 24.064 | **22.27** |

Table 7: Results of different expansion sizes

| $\|V_G\|$ | $\eta$ | $f$ average | | | | | Running time (s) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 1 | 2 | 3 | 4 | 5 |
| 10 | 0.05 | 0.0001 | **0.0000** | 0.0001 | 0.0001 | 0.0002 | 0.14 | 0.09 | 0.09 | 0.08 | **0.07** |
| | 0.1 | 0.0001 | 0.0001 | 0.0001 | **0.0000** | 0.0002 | 0.18 | 0.11 | 0.12 | **0.09** | 0.10 |
| | 0.2 | **0.0137** | 0.0174 | 0.0233 | 0.0220 | 0.0375 | **0.40** | 0.42 | 0.53 | 0.49 | 0.83 |
| 30 | 0.05 | 0.0347 | **0.0297** | 0.0315 | 0.0386 | 0.0322 | 3.36 | 2.19 | 1.89 | 1.95 | **1.78** |
| | 0.1 | 0.1680 | **0.1603** | 0.1712 | 0.1685 | 0.1745 | 4.04 | 2.55 | 2.07 | 1.95 | **1.69** |
| | 0.2 | 0.0088 | 0.0024 | 0.0037 | 0.0025 | **0.0012** | 4.72 | 3.16 | 2.45 | 2.30 | **1.81** |
| 50 | 0.01 | **0.0000** | 0.0000 | 0.0000 | 0.0002 | 0.0002 | 7.71 | 4.34 | 3.33 | 2.92 | **2.41** |
| | 0.05 | **0.0000** | 0.0032 | 0.0025 | 0.0008 | 0.0016 | 11.81 | 6.34 | 4.83 | 4.14 | **3.38** |
| 70 | 0.01 | **0.0002** | 0.0006 | 0.0008 | 0.0011 | 0.0010 | 22.27 | 12.78 | 10.18 | 8.48 | **7.09** |

#### 4.5.1.5 Expansion Size

Table 7 shows the average fitness value and the average running time when different expansion sizes were applied. Since there are classes of graphs with $|V_G|$, we used expansion sizes less than or equal to five. We used MDA reordering scheme and set the stopping criterion threshold as 1%. Correlation coefficients between the average fitness value and the expansion size were also presented. Among the five different sizes, the best results were marked in bold for each class.

Although it takes a long time to run, decreasing the expansion size showed better results for the average fitness value. There was no big difference in the results of sizes 1 and 2, but we chose size 1 because we saw a tendency to improve performance as the overall expansion size decreased.

#### 4.5.1.6 Overall Results

We tested the IHGA using the schemes that showed the best results in the previous experiments. We selected MDA reordering, the stopping criterion with 1% threshold value, and the expansion size value of 1. All of the 20 classes were tested and the results of the IHGA were compared to those of two previous studies which did not apply the incremental process.

Table 8 denotes the overall results. The average fitness value, the standard deviation of the values, the average running time in seconds and the ratio of runs in which an optimal solution has been found are shown in the table. We also performed the Wilcoxon's Signed Rank Test at 5% significance level for the performance comparison of the incremental algorithm with Kim [KM10] and Choi [CYM12].

For the nine relatively difficult classes, the results of the IHGA that were better than those of previous studies are shown in bold. The minimum ratio of finding an optimal solution, in the class that $|V_G|$ is 30 and $\eta$ is 0.05 was dramatically increased from 78.73% to 95.61%. The overall ratios for the 9 difficult classes were also improved from 92.36% to 98.37% and from 96.56% to 99.27% for all classes. The experimental results showed that the well-designed incremental process is helpful to improve the performance of the hybrid genetic algorithm and the IHGA outperforms the previous studies.

Table 8: Overall results of the incremental hybrid genetic algorithm compared to previous genetic algorithms. The results which are better than previous studies are shown in bold.

| $\|V_G\|$ | $\eta$ | **Kim** [KM10] | | | | **Choi** [CYM12] | | | | IHGA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $f$ average | $f$ SD | Time | Ratio | $f$ average | $f$ SD | Time | Ratio | $f$ average | $f$ SD | Time | Ratio | p-value(Kim) | p-value(Choi) |
| 10 | 0.01 | 0.0000 | 0.0000 | 0.03 | 100.00% | 0.0000 | 0.0000 | 0.03 | 100.00% | 0.0000 | 0.0000 | 0.09 | 100.0% | – | – |
| | 0.05 | 0.0020 | 0.0447 | 0.05 | 99.80% | 0.0002 | 0.0042 | 0.06 | 99.82% | **0.0001** | **0.0024** | **0.14** | **99.94%** | $3.214 \times 10^{-5}$ | $7.125 \times 10^{-3}$ |
| | 0.1 | 0.0023 | 0.0479 | 0.08 | 99.77% | 0.0002 | 0.0048 | 0.11 | 99.77% | **0.0001** | **0.0026** | **0.18** | **99.93%** | $5.674 \times 10^{-6}$ | $1.732 \times 10^{-3}$ |
| | 0.2 | 0.4256 | 0.9002 | 0.55 | 79.17% | 0.0427 | 0.0895 | 0.93 | 78.85% | **0.0137** | **0.0549** | **0.40** | **93.49%** | $1.026 \times 10^{-12}$ | $< 2.2 \times 10^{-16}$ |
| 30 | 0.01 | 0.0059 | 0.0766 | 1.13 | 99.41% | 0.0000 | 0.0000 | 0.39 | 100.00% | 0.0000 | 0.0000 | 1.80 | 100.00% | $7.226 \times 10^{-15}$ | – |
| | 0.05 | 5.0088 | 7.0046 | 6.60 | 60.96% | 0.2412 | 0.5020 | 7.09 | 78.73% | **0.0347** | **0.1818** | **3.36** | **95.61%** | $< 2.2 \times 10^{-16}$ | $< 2.2 \times 10^{-16}$ |
| | 0.1 | 5.9865 | 16.2949 | 4.16 | 87.99% | 0.4723 | 1.4629 | 6.31 | 90.43% | **0.1680** | **0.8954** | **4.04** | **96.56%** | $< 2.2 \times 10^{-16}$ | $< 2.2 \times 10^{-16}$ |
| | 0.2 | 0.0226 | 1.5980 | 1.04 | 99.98% | 0.0047 | 0.2350 | 2.52 | 99.96% | 0.0088 | 0.3307 | 4.72 | 99.93% | 0.7928 | 0.8174 |
| 50 | 0.01 | 2.2322 | 1.3002 | 15.19 | 4.80% | 0.0043 | 0.0203 | 7.29 | 95.69% | **0.0000** | **0.0000** | **7.71** | **100.00%** | $< 2.2 \times 10^{-16}$ | $< 2.2 \times 10^{-16}$ |
| | 0.05 | 8.6474 | 25.8336 | 10.50 | 89.78% | 0.0015 | 0.1093 | 3.51 | 99.98% | **0.0000** | **0.0000** | **11.81** | **100.00%** | $< 2.2 \times 10^{-16}$ | $7.865 \times 10^{-3}$ |
| | 0.1 | 0.0875 | 4.3795 | 3.72 | 99.96% | 0.0000 | 0.0000 | 3.65 | 100.00% | 0.0021 | 0.2070 | 13.95 | 99.99% | $2.274 \times 10^{-3}$ | 0.8414 |
| | 0.2 | 0.0000 | 0.0000 | 1.51 | 100.00% | 0.0000 | 0.0000 | 3.86 | 100.00% | 0.0000 | 0.0000 | 18.08 | 100.00% | – | – |
| 70 | 0.01 | 16.0792 | 5.2860 | 32.32 | 0.00% | 0.0126 | 0.0351 | 14.48 | 88.01% | **0.0002** | **0.0041** | **22.27** | **99.86%** | $< 2.2 \times 10^{-16}$ | $< 2.2 \times 10^{-16}$ |
| | 0.05 | 0.9251 | 14.4306 | 9.41 | 99.59% | 0.0000 | 0.0000 | 2.76 | 100.00% | 0.0000 | 0.0000 | 28.88 | 100.00% | – | $7.299 \times 10^{-11}$ |
| | 0.1 | 0.0000 | 0.0000 | 3.49 | 100.00% | 0.0000 | 0.0000 | 3.92 | 100.00% | 0.0000 | 0.0000 | 34.57 | 100.00% | – | – |
| | 0.2 | 0.0000 | 0.0000 | 2.37 | 100.00% | 0.0000 | 0.0000 | 6.52 | 100.00% | 0.0000 | 0.0000 | 45.12 | 100.00% | – | – |
| 90 | 0.01 | 8.9344 | 8.0415 | 52.60 | 14.52% | 0.0000 | 0.0000 | 6.75 | 100.00% | 0.0000 | 0.0000 | 46.09 | 100.00% | $< 2.2 \times 10^{-16}$ | – |
| | 0.05 | 0.0000 | 0.0000 | 4.93 | 100.00% | 0.0000 | 0.0000 | 3.52 | 100.00% | 0.0000 | 0.0000 | 56.26 | 100.00% | – | – |
| | 0.1 | 0.0000 | 0.0000 | 3.63 | 100.00% | 0.0000 | 0.0000 | 4.55 | 100.00% | 0.0000 | 0.0000 | 68.98 | 100.00% | – | – |
| | 0.2 | 0.0000 | 0.0000 | 3.43 | 100.00% | 0.0000 | 0.0000 | 7.67 | 100.00% | 0.0000 | 0.0000 | 91.07 | 100.00% | – | – |

One disadvantage of the IHGA revealed through the experimental results was that it takes longer time to execute than the typical GA, **Choi**. However, if we look at the results by class, we can see that the running times of typical GA were short for easy classes, but for the difficult classes, the running times of the IHGA were shorter. Therefore, we performed two additional experiments to analyze the running time of the **Choi** and the IHGA. The results are shown in Table 9.

First, we set 1% threshold value for the stopping criterion and counted the number of generations the two algorithms would terminate. In most classes, both algorithms found the optimal solution in a very short generations. Since even if the subproblems are easy, at least one generation should be executed for all subproblems. For this reason, the running time of the IHGA seems to be longer. However, as the number of generations increases for difficult classes, there was a different tendency in running time. For difficult classes where the number of generations of the typical GA exceeded 30, such as the class (30, 0.05), the IHGA found the optimal solution with a relatively small number of generations. In this case, the IHGA is more effective in terms of running time, and the results reflecting these are shown in Table 8. This trend can be confirmed by the running time obtained when $TH$ was set to $\infty$ so that the both algorithms run the same number of generations. Thus, although there are some penalties for running time for easy problems, we can conclude that the more difficult the problem is, the IHGA shows better performance in terms of accuracy and running time than the previous work.

Table 9: Running time analysis between the conventional hybrid GA and the IHGA

| $\|V_G\|$ | $\eta$ | The number of generations (when $TH = 1$) | | Running time (s) (when $TH = \infty$) | |
|---|---|---|---|---|---|
| | | **Choi** [CYM12] | IHGA | **Choi** [CYM12] | IHGA |
| 10 | 0.01 | 1.00 | 10.00 | 0.88 | 0.42 |
| | 0.05 | 3.11 | 10.37 | 1.21 | 0.61 |
| | 0.1 | 5.03 | 10.47 | 1.47 | 0.73 |
| | 0.2 | 42.64 | 21.38 | 2.10 | 0.97 |
| 30 | 0.01 | 4.13 | 30.02 | 6.16 | 3.79 |
| | 0.05 | 67.15 | 40.67 | 10.90 | 6.53 |
| | 0.1 | 37.34 | 38.21 | 14.36 | 7.30 |
| | 0.2 | 7.01 | 31.72 | 17.28 | 8.65 |
| 50 | 0.01 | 41.49 | 50.14 | 17.06 | 11.11 |
| | 0.05 | 8.53 | 51.07 | 29.28 | 16.89 |
| | 0.1 | 4.75 | 52.39 | 34.11 | 19.29 |
| | 0.2 | 2.02 | 51.66 | 46.40 | 24.78 |
| 70 | 0.01 | 34.92 | 70.33 | 40.71 | 30.80 |
| | 0.05 | 1.00 | 70.00 | 51.38 | 39.02 |
| | 0.1 | 1.00 | 70.00 | 59.25 | 46.42 |
| | 0.2 | 1.00 | 70.00 | 83.56 | 60.85 |
| 90 | 0.01 | 6.42 | 90.00 | 72.46 | 52.82 |
| | 0.05 | 1.00 | 90.00 | 77.13 | 63.38 |
| | 0.1 | 1.00 | 90.00 | 94.26 | 77.68 |
| | 0.2 | 1.00 | 90.00 | 128.24 | 102.89 |

#### 4.5.1.7 Experiments for structured graphs

In this section, we experimented further with structured graphs with various types and sizes to identify the limitations of the incremental algorithm.

Table 10: Performance of IHGA on structured graphs

(a) Circular graphs with different $d$

| $\|V_G\|$ | $d=1$ | | $d=5$ | | $d=10$ | | $d=15$ | |
|---|---|---|---|---|---|---|---|---|
| | f average | Ratio | f average | Ratio | f average | Ratio | f average | Ratio |
| 10 | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% |
| 30 | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% |
| 50 | 0.0000 | 100.00% | 0.0704 | 57.30% | 0.0002 | 99.80% | 0.0002 | 99.80% |
| 70 | 0.0290 | 75.40% | 0.1711 | 49.90% | 0.0128 | 88.40% | 0.0251 | 76.70% |
| 90 | 0.2941 | 3.70% | 0.4363 | 59.10% | 0.0000 | 100.00% | 0.0000 | 100.00% |

(b) Complete $d$-ary tree with diffrerent $d$

| $\|V_G\|$ | $d=1$ | | $d=2$ | | $d=3$ | | $d=4$ | |
|---|---|---|---|---|---|---|---|---|
| | f average | Ratio | f average | Ratio | f average | Ratio | f average | Ratio |
| 10 | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% |
| 30 | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% |
| 50 | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% | 0.0000 | 100.00% |
| 70 | 0.0243 | 79.30% | 0.0810 | 42.60% | 0.0005 | 99.60% | 0.0000 | 100.00% |
| 90 | 0.4304 | 0.40% | 0.1010 | 68.50% | 0.0000 | 100.00% | 0.0000 | 100.00% |

- Circle

  For each vertex $i$, there are edges going to vertices $i+1, i+2, \ldots, i+d$. We experimented with $d = 1, 5, 10$, and 15. In the case of $d = 1$, it is a circular graph in the form of a circle.

- Tree

  This is a graph in the form of complete $d$-ary tree. For each vertex $i$, there are edges going to vertex $i \times d + 1, i \times d + 2, \ldots, i \times d + d$. We experimented with $d = 1, 2, 3, 4$.

Table 11: Statistics of 7 biological networks

| | Name | $\|V\|$ | $\|E\|$ | $\eta$ | $d_{max}$ | $d_{avg}$ | Assortativity |
|---|---|---|---|---|---|---|---|
| $H_1$ | celegans | 453 | 2025 | 0.0197796 | 237 | 8 | –0.225821 |
| $H_2$ | diseasome | 516 | 1188 | 0.00894107 | 50 | 4 | 0.0666456 |
| $H_3$ | SC-TC | 636 | 3959 | 0.0196058 | 66 | 12 | 0.921112 |
| $H_4$ | DM-LC | 658 | 1129 | 0.00522315 | 50 | 3 | –0.121817 |
| $H_5$ | CE-GT | 924 | 3239 | 0.00759569 | 151 | 7 | –0.159339 |
| $H_6$ | grid-mouse | 1455 | 3272 | 0.00311463 | 222 | 4 | –0.153014 |
| $H_7$ | yeast | 1458 | 1948 | 0.00183401 | 56 | 2 | –0.209541 |

As can be seen in Table 10, we can observe that the performance of the algorithm is degraded when the algorithm needs to be matched up to the correct permutation in a graph with a simple but redundant structures.

### 4.5.2 Real World Data

#### 4.5.2.1 Dataset and Evaluation

In previous experiments, we tested several design schemes to verify the effectiveness of the incremental process and compared the performance of the IHGA with other algorithms. In this subsection, we tested the performance and scalability of the IHGA for real world data consisting of large graphs. We collected 7 kinds of biological networks from Network Repository[1] [RA15], which have hundreds to thousand number of vertices. Table 11 shows the statistics of these biological networks.

The one whole network was set to $H$. And for one problem case of $H$ and $|V_G|$, we independently generated 10 problem instances by sampling $G$ in the same way as Subsection 4.5.1. We chose 50, 100 as $|V_G|$ and con-

[1]Network repository: biological networks, http://networkrepository.com/bio.php

ducted 100 runs for each instance using the IHGA and averaged the results of all instance in each problem case. We set MDA reordering for vertex ordering scheme and 1% threshold value for stopping criterion. And we measured the average fitness value and running time with 5 different expansion size, $S$, from 1 to 5.

#### 4.5.2.2 Experimental results on biological networks

First of all, in terms of the average fitness value, the IHGA found the optimal solution in most cases, even though the graphs $G$ and $H$ become larger. However, in terms of running time, increasing the expansion size shortened the time by 3 to 4 times. The instances of previous synthetic data were relatively easy to solve, so that reducing the expansion size and exploiting the subspace in detail improved the quality of solution. However, in this experiment, we found that it is more efficient to set the search space larger for each subproblem because the size of instances are relatively large and the size of problem space is also increased accordingly. Although it took a long time to execute as $|V_G|$ grows in size, but we can shorten the running time while maintaining the quality of solutions well by increasing the expansion size. As a result, we have confirmed that our algorithm works well in real world data and has scalability.

Table 12: Results on biological networks. $|V_G| = \{50, 100\}$ and expansion size $S = \{1, 2, 3, 4, 5\}$

(a) $f$ average

| $\|V_G\|$ | $S$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_5$ | $H_6$ | $H_7$ |
|---|---|---|---|---|---|---|---|---|
| 50 | 1 | 0.00 | | | | | | |
| | 2 | | | | | | | |
| | 3 | | | | | | | |
| | 4 | | | | | | | |
| | 5 | | | | | | | |
| 100 | 1 | $8.34 \times 10^{-2}$ | $5.31 \times 10^{-3}$ | $\mathbf{6.29 \times 10^{-5}}$ | 0.00 | 0.00 | 0.00 | 0.00 |
| | 2 | $8.01 \times 10^{-2}$ | $5.81 \times 10^{-3}$ | $\mathbf{3.14 \times 10^{-5}}$ | 0.00 | 0.00 | 0.00 | 0.00 |
| | 3 | $8.27 \times 10^{-2}$ | $5.54 \times 10^{-3}$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 4 | $7.79 \times 10^{-2}$ | $6.36 \times 10^{-3}$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | 5 | $8.24 \times 10^{-2}$ | $5.70 \times 10^{-3}$ | $\mathbf{3.14 \times 10^{-5}}$ | 0.00 | 0.00 | 0.00 | 0.00 |

(b) Running time (s)

| $\|V_G\|$ | $S$ | $H_1$ | $H_2$ | $H_3$ | $H_4$ | $H_5$ | $H_6$ | $H_7$ |
|---|---|---|---|---|---|---|---|---|
| 50 | 1 | 44.49 | 43.67 | 59.08 | 55.67 | 88.93 | 126.58 | 138.82 |
| | 2 | 25.16 | 24.34 | 33.30 | 30.65 | 49.42 | 66.87 | 73.58 |
| | 3 | 18.98 | 18.26 | 24.95 | 22.93 | 37.07 | 48.98 | 54.17 |
| | 4 | 15.88 | 15.06 | 20.55 | 18.93 | 30.68 | 40.04 | 44.29 |
| | 5 | 12.83 | 11.67 | 15.84 | 14.81 | 23.94 | 30.78 | 34.16 |
| 100 | 1 | 848.50 | 485.57 | 458.57 | 420.22 | 761.45 | 991.44 | 1077.79 |
| | 2 | 738.14 | 359.40 | 261.41 | 233.04 | 457.14 | 527.73 | 577.75 |
| | 3 | 717.83 | 313.23 | 194.85 | 173.07 | 360.90 | 388.41 | 427.29 |
| | 4 | 677.70 | 288.77 | 148.49 | 130.57 | 288.89 | 289.60 | 321.39 |
| | 5 | 686.64 | 272.82 | 125.71 | 108.24 | 258.06 | 241.78 | 267.37 |

#### 4.5.2.3 Experimental results with different genetic parameters

We observed the relation among different genetic parameters with the performance of the IHGA through additional experiments. For the biological networks $H_2$, we measured the ratio of runs in which the optimal solution has been found and the average running time with three different parameter settings based on the previous experiment.

First, we set the population size to 100, 200, and $|V_H|$ and generate half of population size offspring for each generation. Table 13 shows the result. We found that increasing the population size and hence a number of offspring resulted in a slight improvement in average fitness value, but on the contrary, it needed a much longer run time.

In the second, performance of various settings was observed while changing a number of generations to 100, 200, and $|V_H|$. Table ref tab: gen shows the performance of IHGA according to a number of generation changes. In the first three settings, we confirmed that reducing the number of generations shortens the running time, but reduces the performance as well. In the last three cases, we recognized that there is a difference in the performance improvement of IHGA according to population change and

Table 13: Comparative analysis of IHGA with different population size

| | $\|V_H\|$ | Population | offspring | Expansion size | Generation | Time(s) | Ratio |
|---|---|---|---|---|---|---|---|
| $H_2$ | 516 | 100 | 50 | 5 | 516 | 588.82 | 84.50% |
| | | 200 | 100 | | | 616.55 | 87.00% |
| | | 516 | 258 | | | 1345.30 | 90.00% |

Table 14: Comparative analysis of IHGA with different a number of generations

| | $\|V_H\|$ | Population | offspring | Expansion size | Generation | Time(s) | Ratio |
|---|---|---|---|---|---|---|---|
| $H_2$ | 516 | 100 | 50 | 5 | 516 | 588.82 | 84.50% |
| | | 100 | 50 | 12 | 200 | 218.33 | 74.00% |
| | | 100 | 50 | 25 | 100 | 138.41 | 65.30% |
| | | 100 | 50 | 25 | 200 | 206.76 | 73.80% |
| | | 200 | 100 | 25 | 100 | 262.09 | 67.30% |

Table 15: Comparative analysis of IHGA with different parameters which generate the same number of solutions during the whole execution

| | $\|V_H\|$ | Population | offspring | Expansion size | Generation | Time(s) | Ratio |
|---|---|---|---|---|---|---|---|
| $H_2$ | 516 | 100 | 50 | 5 | 516 | 588.82 | 84.50% |
| | | 200 | 100 | 10 | 258 | 411.79 | 81.90% |
| | | 258 | 129 | 12 | 200 | 510.38 | 77.60% |
| | | 516 | 258 | 25 | 100 | 618.87 | 70.30% |

generation change. Even if we create the same number of offspring during the whole execution, increasing a number of generation results in more performance improvement than increasing a population size.

To clarify this tendency, we observed performance by changing an expansion size and a number of generations to produce the same number of solutions during the whole incremental algorithm. As shown in Table 15, we could more reliably confirm the characteristics of the IHGA found in Table 14 through experiments with settings that produce the same number of solutions. Rather than expanding each subproblem to a larger scale through a large-size population, it has been shown that expanding the subproblem to a smaller scale and evolving the population gradually over generations will improve performance.

# Chapter 5

# Conclusion

In this thesis, we investigated methods to improve the search capability of a genetic algorithm for the subgraph isomorphism problem. In general, new operators or local optimization algorithms are designed to directly improve the search capability of an search algorithm. Instead of this, we claimed that the search capability of an algorithm can be improved by indirect methods; One is changing a fitness function that appropriates the property of the problem and transforms the fitness landscape more globally convex. The other is reforming the search strategy that divides a large problem into multiple subproblems and extends solutions one by one. Applying these methods, we showed that the proposed incremental hybrid genetic algorithm exceeds the existing algorithms through various experiments for the subgraph isomorphism problem.

First, it is verified that a multi-objective fitness function which conforms to the degree property of the subgraph isomorphism transforms a fitness landscape globally convex and facilitates exploiting the space by the local optimization algorithm. We revealed that the fitness function used in previous studies does not reflect the degree constraint of the subgraph isomorphism properly. To relax this limitation, we designed a new function and proposed a multi-objective fitness function by linear combination with the existing function. Experiments in combination with the two-exchange

local heuristic showed that the existing function does not reduce the number of vertices which are incorrectly matched over steps, while the new fitness function satisfies the degree constraint and finds better solutions based on this. For each local optimum, the correlation coefficients between the average distance to the other optima and the fitness value were calculated, and the results of a multi-objective function showed stronger positive correlation than that of function previously used. From these results, it infers that the fitness landscape is transformed more globally convex. And when applying to the hybrid genetic algorithm, it was confirmed that a multi-objective fitness function shows better performance than the existing function.

Second, we described the incremental genetic algorithm as a new search strategy for in the subgraph isomorphism problem. The incremental algorithm begins with decomposing the original problem into a sequence of consecutive subproblems that satisfies the optimal substructure property. The each subproblem is solved by the hybrid genetic algorithm and the solutions obtained are extended as initial solutions for the next subproblem. The subproblems are sequentially solved in this way, and the solutions of the original problem are finally obtained. We noticed that designing a sequence of subproblems is most important in the performance of the incremental algorithm, and introduced several design schemes. Vertex reordering determines the entire search path of the incremental genetic algorithm. Experimental results showed that the maximal adjacency-degree reordering scheme combining adjacency information and degree information has the best performance. Stopping criterion and expansion size determine the size of the problem space in which the hybrid genetic algorithm explores and ex-

ploits in each subproblem. The schemes to preserve the diversity of solutions in the incremental process showed the best in the experiments. The effectiveness of the incremental process were proved by comparing the performance between extending the solutions from the previous subproblem and partially initializing solutions. The reason why the incremental algorithm more running time than the conventional genetic algorithm is that each subproblem must be solved at least on one generation, regardless of how easy the subproblem is. Based on this analysis, we showed through experiments that the well-designed incremental hybrid genetic algorithm outperforms the previous proposed algorithms. Furthermore, it was confirmed that the proposed algorithm has scalability through the experiments based on the real world data with a large graph size.

Although we traced the several design schemes for the incremental approach, there are still other issues that have to be figured out for better performance. As shown in the experiments, our methods for building a sequence of subproblems and tuning parameters are manually designed. In the future, we are considering the adaptive features which are able to set parameters automatically and the dynamic configurations which can change the schemes in the process of solving the subproblems. Also, we hope to apply this algorithm in the real world application with the various size and types of the graphs.

# Bibliography


[AW10] Charu C Aggarwal and Haixun Wang. Graph data management and mining: A survey of algorithms and applications. In *Managing and mining graph data*, pages 13–68. Springer, 2010.

[BBK11] Gözde Bakırlı, Derya Birant, and Alp Kut. An incremental genetic algorithm for classification and sensitivity analysis of its parameters. *Expert Systems with Applications*, 38(3):2609 – 2620, 2011.

[BJWG94] Robert D Brown, Gareth Jones, Peter Willett, and Robert C Glen. Matching two-dimensional chemical graphs using genetic algorithms. *Journal of Chemical Information and Computer Sciences*, 34(1):63–70, 1994.

[BKM94] Kenneth D. Boese, Andrew B. Kahng, and Sudhakar Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 16(2):101 – 113, 1994.

[BY86] Egon Balas and Chang Sung Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054–1068, 1986.

[CFSV04a] D. CONTE, P. FOGGIA, C. SANSONE, and M. VENTO. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03):265–298, 2004.

[CFSV04b] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, Oct 2004.

[CFV07] Donatello Conte, Pasquale Foggia, and Mario Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1):99–143, 2007.

[CKM14] HyukGeun Choi, JinHyun Kim, and Byung-Ro Moon. A hybrid incremental genetic algorithm for subgraph isomorphism problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 445–452, New York, NY, USA, 2014. ACM.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[CYM12] Jaeun Choi, Yourim Yoon, and Byung-Ro Moon. An efficient genetic algorithm for subgraph isomorphism. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 361–368, New York, NY, USA, 2012. ACM.

[Die92] Gröger Hans Dietmar. On the randomized complexity of monotone graph properties. *Acta Cybern.*, 10(3):119–127, October 1992.

[dlHJ+13] Colin de la Higuera, Jean-Christophe Janodet, Émilie Samuel, Guillaume Damiand, and Christine Solnon. Polynomial algorithms for open plane graph and subgraph isomorphisms. *Theoretical Computer Science*, 498:76 – 99, 2013.

[DPBT99] Paul J Durand, Rohit Pasari, Johnnie W Baker, and Chun-che Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17):1–16, 1999.

[ES+03] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[FC15] M. M. Farahani and S. K. Chaharsoughi. A genetic and iterative local search algorithm for solving subgraph isomorphism problem. In *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, pages 1–6, March 2015.

[FKZ01] Hubert Fröhlich, Andrej Košir, and Baldomir Zajc. Optimization of fpga configurations using parallel genetic algorithm. *Information Sciences*, 133(3-4):195–219, 2001.

[FSV01] Pasquale Foggia, Carlo Sansone, and Mario Vento. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. In *Proc. of the 3rd IAPR TC-15 International Workshop on Graph-based Representations*, pages 176–187, 2001.

[Gal06] Brian Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[GJ02] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[GXTL10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129, Feb 2010.

[HKM06] Inwook Hwang, Yong-Hyuk Kim, and Byung-Ro Moon. Multi-attractor gene reordering for graph bisection. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1209–1216, New York, NY, USA, 2006. ACM.

[Hol75] John Holland. Adaptation in natural and artificial systems: an introductory analysis with application to biology. *Control and artificial intelligence*, 1975.

[HS08] Huahai He and Ambuj K. Singh. Graphs-at-a-time: Query language and access methods for graph databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 405–418, New York, NY, USA, 2008. ACM.

[KCYM16] Jinhyun Kim, HyukGeun Choi, Hansang Yun, and Byung-Ro Moon. Measuring source code similarity by finding similar subgraph with an incremental genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 925–932, New York, NY, USA, 2016. ACM.

[KM10] Keehyung Kim and Byung-Ro Moon. Malware detection based on dependency graph using hybrid genetic algorithm. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 1211–1218, New York, NY, USA, 2010. ACM.

[LCC16] Zuqing Li, Bernard Chen, and Dongsheng Che. Solving the subgraph isomorphism problem using simulated annealing and evolutionary algorithms. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 293. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.

[LHKL12] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proc. VLDB Endow.*, 6(2):133–144, December 2012.

[MAEF06] Nashat Mansour, Mohamad Awad, and Khaled El-Fakih. Incremental genetic algorithm. *The International Arab Journal of Information Technology*, 3(1):42–47, 2006.

[MB00] B. T. Messmer and H. Bunke. Efficient subgraph isomorphism detection: a decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, Mar 2000.

[McG82] James J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.

[OSH87] IM Oliver, DJd Smith, and John RC Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[RA15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Sch88] Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312 – 323, 1988.

[SWG02] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 39–52, New York, NY, USA, 2002. ACM.

[SZLY08] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. *Proc. VLDB Endow.*, 1(1):364–375, August 2008.

[Ull76] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, January 1976.

[VL91] Gregor Von Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Proceedings of the 4th International Conference on Genetic Algorithms*, ICCA '91, 1991.

[VN10] P Vivekanandan and R Nedunchezhian. A new incremental genetic algorithm based classification model to mine data with concept drift. *Journal of Theoretical & Applied Information Technology*, 21(1), 2010.

[WG94] Markus Wagener and Johann Gasteiger. The determination of maximum common substructures by a genetic algorithm: Application in synthesis design and for the structural analysis of biological activity. *Angewandte Chemie International Edition*, 33(11):1189–1192, 1994.

[WM97] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

[WYJ+04] A. S. Wu, H. Yu, S. Jin, K. C. Lin, and G. Schiavone. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):824–834, Sept 2004.

[ZH10] Peixiang Zhao and Jiawei Han. On graph query optimization in large networks. *Proc. VLDB Endow.*, 3(1-2):340–351, September 2010.

[ZLY09] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: Distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 192–203, New York, NY, USA, 2009. ACM.

[ZWL+11] Q. Zhong, Z. Wu, L. Lin, Y. Zhang, and J. Zhang. Computing resources assignment in rtds simulators with subgraph isomorphism based on genetic algorithm. In *2011 4th International*

*Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT)*, pages 1144–1149, July 2011.

# 국문 초록

그래프는 객체들의 관계를 표현하는 가장 대표적인 자료구조이고, 데이터가 그래프 형태로 표현되는 많은 연구분야에서 발생하는 핵심 문제들 중 하나가 바로 그래프 패턴 매칭이다. 그래프 패턴 매칭은 정점이나 간선들의 정보를 이용한 시멘틱 기반의 방법으로 정의할 수도 있지만, 일반적으로는 정점과 간선간의 관계만을 이용해 구조적으로 정의하고 이러한 패턴매칭은 부분그래프 동형사상으로 표현된다.

그동안 부분그래프 동형사상 문제를 풀기 위해 제안된 알고리즘들은 크게 두 가지 형태로 분류된다. 첫번째는 재귀적 퇴각검색 알고리즘을 기반으로 존재하는 모든 해를 정확하게 찾아내는 방법이다. 다만, 부분그래프 동형사상 문제는 대표적인 NP-완비군의 문제 중 하나이기 때문에 모든 순열을 하나씩 탐색하는 경우 수행시간이 문제의 크기에 따라 기하급수적으로 늘어나게 된다. 두번째는 유전 알고리즘을 비롯한 메타휴리스틱 알고리즘 기반의 근사적인 방법이다. 이들은 합리적인 시간 내에 좋은 품질의 해들을 찾아내지만 대부분의 알고리즘이 그 크고 복잡한 문제공간 전체를 다룰 수 있을 만큼의 탐색능력을 갖추지는 못하였다.

연산자나 지역 휴리스틱을 개선하여 알고리즘의 공간탐색능력을 직접적으로 향상시킬수도 있겠지만, 적합도 함수를 변경하거나 탐색전략 변경을 통해서도 크게 성능을 개선할 수 있다. 만약 원래 문제를 메타 휴리스틱의 탐색능력에 적합한 크기의 부분문제로 분할하고, 이 부분문제를 단계적으로 풀어간다면 보다 효율적으로 문제를 해결할 수 있다. 또한, 적합도 함수를 변경하여 공간을 보다 단순한 형태로 변환시킨다면 그 효

과가 훨씬 더 커질것이다.

본 논문에서는 부분그래프 동형사상 문제가 이루는 문제공간의 특성을 분석하고 이에 어울리는 적합도 함수와 탐색전략을 바탕으로, 이 문제를 효율적으로 풀기 위한 유전알고리즘을 제안한다. 첫번째로, 부분그래프 동형사상 문제에 어울리는 새로운 적합도 함수를 소개하고, 연산자와 함께 생성되는 적합도 공간이 어떠한 형태로 변형되는지를 살펴본다. 우선, 기존 연구들에서 사용한 적합도 함수가 가지고 있던 문제점들을 검토하고 이를 해결하기 위해 부분그래프 동형사상의 정점의 차수 조건을 반영한 새로운 함수를 설계해서 기존의 함수와 결합한 다목적 적합도 함수를 제안한다. 이후, 실험을 통해서 지역 최적화 알고리즘과 결합했을 때 적합도 값들의 변화과정을 통해 새로운 적합도 함수의 특징들을 분석하고 지역최적점들을 모아 적합도 함수와 해들의 평균거리를 이용한 상관관계를 통해 제안한 적합도 함수가 그리는 문제공간이 기존의 문제공간을 어떤 식으로 변형시키는지를 설명한다. 제안한 다목적 적합도 함수를 혼합형 유전알고리즘에 적용한 결과를 기존 연구들의 결과들과 비교하여 제안한 다목적 적합도 함수가 유전알고리즘의 문제공간탐색과 최적화에 얼마나 도움을 주는지를 확인한다.

두번째로, 새롭게 설계된 문제공간을 효율적으로 탐색하기 위한 전략으로 점진적 유전 알고리즘을 소개하고 각 설계요소들이 알고리즘의 수행과정과 성능에 어떻게 반영되는지를 알아본다. 우선, 점진적 유전알고리즘에서 원 문제를 최적 부분구조를 갖는 일련의 연속적인 부분문제들로 분할한 후 각 부분문제를 혼합형 유전알고리즘을 통해 풀고 얻어진 해들을 확장하여 다음 부분문제의 초기해로 사용하는 방법을 설명하고, 이러한 과정을 순차적으로 적용하여 작은 부분문제의 해를 원래 문제의 해로 발전시켜 원 문제의 답을 얻는 과정을 보인다. 이후, 점진적 유전

알고리즘을 진행하는 과정에서 원래의 문제를 분할하는 방법과 부분문제들의 연속성을 설정하는 부분이 알고리즘 전체 성능에 어느 정도 영향을 미치는지를 실험을 통해 분석한다. 최종적으로 랜덤그래프에 대해서 제안한 점진적 혼합 유전 알고리즘의 성능과 기존의 혼합형 유전알고리즘의 성능을 비교분석하고, 기존의 알고리즘들로는 불가능했던 사이즈가 큰 실제 데이터들에 대해서도 좋은 성능을 보임으로써 확장성까지 갖춘 것을 보여준다.

# 감사의 글

오랜 시간을 관악에서 지내는 동안 많은 사람들을 만나고 함께하며 헤어지게 되었습니다. 이 모든 분들의 응원과 도움으로 이렇게 학위를 받게 된 것 같아 이 자리를 빌어 감사의 말씀을 드립니다.

우선, 지금까지 저를 지도해주신 지도교수 문병로 교수님께 감사를 드립니다. 제 학위논문을 심사해주신 신영길 교수님, 엄현상 교수님, 김용혁 교수님 그리고 정순철 박사님께도 감사드립니다. 다음으로 저와 함께 논문을 작성한 윤유림 교수님, 김진현 박사님께 깊은 감사의 말씀을 드리고 싶습니다. 이분들의 도움을 받아 학위논문을 무사히 완성할 수 있었던 것 같습니다. 또한, 제 논문작성에 많은 도움이 되었던 최자은 석사님께도 감사드립니다.

함께 연구실 생활을 한 많은 선후배님들을 만나 즐겁고 힘들었지만 행복했던 시간도 생각납니다. 먼저 떠난 나의 동기들 조일룡, 손유민, 김기형이 있어 연구실 첫 생활이 참 즐거웠습니다. 저에게 처음 스터디와 논문쓰기를 지도해주신 오진수 박사님, 저를 믿고 많은 도움을 주셨던 운경목 박사님, 저에게 즐거움과 많은 기회를 주었던 정찬주 박사님, 여러 가지 모습을 갖추신 존경스러운 이민기 박사님, 그리고 기계학습과 데이터과학 선생님이었던 하성주 박사님께 감사를 드립니다. 본인의 연구에 매진하는 모습이 부러웠던 윤한상과 이상엽, 프로젝트를 하면서 함께 고생했던 임희창, 연구실 분위기를 항상 밝게 만들어주는 엄승현, 육지은, 연구실 생활의 마지막을 함께 한 송예지와 김창겸, 이희재, 지승근까지 모두 감사합니다. 그리고 먼저 연구실을 떠나신 재영이형, 수상, 상철, 승현, 지훈, 강산, 일규 그리고 함께 나가는 채헌이까지 모두모두 함께해서

즐거웠습니다. 무엇보다도 저와 옵투스에서 동고동락한 이승규 박사님, 문승현 선배님, 동료 하명훈에게 깊은 감사의 말씀을 전합니다.

그리고 지금까지 저의 학위기간동안 함께해주시고 한결같이 응원해주신 부모님과 가족들에게 정말 깊은 감사를 드립니다. 멀리 떨어져 계시지만 대전가족분들의 응원도 제게 큰 힘이 되었습니다. 또한, 학위 과정 중 인연이 닿아 많은 도움을 주신 약천스님께 큰 감사의 말씀 드립니다.

마지막으로 사랑하는 아내 윤하정와 아들 최태영, 이 두 사람이 없었다면 여기까지 오지 못했을 것 같습니다. 두 사람과 함께 이 학위수여의 기쁨을 함께하겠습니다.